UNIVERSITY OF OSLO
Department of Informatics

Investigating Resource Consumption of Techniques for Conveying Visual Information in Wireless Networks of Mobile Devices

Master thesis

Anders Berger
Vedahl Ullnæss

June 8, 2012

Investigating Resource Consumption of Techniques for Conveying Visual Information in Wireless Networks of Mobile Devices

Anders Ullnæss

Spring 2012

Abstract

In an emergency and rescue (ER) scenario, e.g. in the aftermath of an earthquake, the regular communication infrastructure might not be available. Modern mobile devices can still, however, form a network between themselves. Such a network is commonly referred to as a Mobile Ad-hoc Network (MANET) and can be utilized for communication purposes. The nodes in a MANET are typically handheld devices with limited power and bandwidth. Due to the mobility and limited communication range of the devices, links between them may go up and down, A network with these properties is called a disruptive MANET. In a disruptive MANET there is no guarantee that we have a complete route from source to destination up at any one time. We therefore utilize delay tolerant networking techniques, such as a store-carry-forward scheme to ensure eventual delivery of packets.

We picture a scenario with two main locations and associated network partitions, namely the incident site and the off site Command and Control Center (CCC). Rescue workers at the incident site use head-mounted cameras to capture video containing information that needs to be transmitted to the personnel at the CCC. Transmission, being one of the main sources of power consumption can put a large toll on the network and lead to the depletion of the batteries of key devices at the emergency site.

Our goal is to reduce resource usage when conveying information based on video from an incident site to the CCC in a disruptive MANET. By extracting the information we are interested in from the video and discarding irrelevant information we reduce the power consumption and required bandwidth. Additionaly, we allow the most important information to reach the destination more quickly. We analyze the application scenario and come up with different approaches for doing this, one of them being to select key frames to represent whole video segments and transmit these frames as opposed to the whole video. The personnel at the receiving end can then decide which segments to receive first, based on what they see in the key frames. Another approach is transferring panoramic images, which are the results of stitching multiple overlapping images together to help give the receiver an overview of the site where the footage was taken. We propose a possible application for the workers at the CCC utilizing these and other approaches to give the personnel a best possible overview of the situation at the incident site.

We circle out one of the components of the CCC application, the image stitching component, and conduct an experiment to investigate the resource usage and power consumption of different approaches to utilizing image stitching. The results of this experiment shows that image stitching can help reduce the power consumption, amount of data transmitted and time spent compared to the transmission of video.

Acknowledgements

I want to begin by thanking my supervisors Morten Lindeberg and Thomas Plagemann from the Distributed Multimedia Systems (DMMS) research group for their guidance, support, suggestions and constructive criticism while writing this thesis. I would also like to thank Asbjørn Brekke and Fritz Albregtsen from the Digital Signal Processing and Image Analysis (DSB) research group for their input and assistance in the image analysis related topics of my work. Further on, I want to thank Olav Stanly Kyrvestad at the Nanoelectronics research group for letting me use their lab for my experiment and master student Tor Eivind Bjørnstad from the same group for helping me set it up. I want to thank Hans Vatne Hansen at the DMMS group for answering my questions regarding Android programming, porting of libraries and applications to the Android device and sometimes just listening while I figured out what the problem was myself. Finally I would like to thank my fellow master student, Tomas Gryczon at the DMMS group, for assisting me with figure creation and proof reading and providing a second opinion when I was unsure. Without the help of these individuals, this work would never have been possible.

Anders Ullnæss
University of Oslo
June, 2012

Contents

List of Figures

List of Tables

List of Algorithms

List of abbreviations

ADB	Android Debug Bridge
AODV	Ad-hoc On-demand Distance Vector routing
AP	Access Point
API	Application programming interface
CCC	Command and Control Center
CPU	Central Processing Unit
DMMS	Distributed Mulitmedia Systems
DTN	Delay Tolerant Network
DTS	Delay Tolerant Streaming
ER	Emergency and Rescue
GOP	Group of Pictures
ICS	Ice Cream Sandwich
ISM	Industrial, Scientific and Medical
JNI	Java Native Interface
JPEG	Joint Photographic Experts Group
MANET	Mobile Ad-hoc Network
mAh	milliampere-hour
MPEG	Moving Picture Experts Group
MPR	Multipoint Relay
OLSR	Optimized Link State Routing
OpenCV	Open Source Computer Vision
RAM	Random Access Memory
RANSAC	Random Sample Consensus
RERR	Route Error
RREP	Route Reply
RREQ	Route Request
SIFT	Scale-Invariant Feature Transform
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WLAN	Wireless Local Area Network

Chapter 1

Introduction

This thesis is done as part of the research of the Distributed Multimedia Systems (DMMS) research group at the Institute of Informatics at the University of Oslo. It is part of a project known as the DT-Stream project [24]. The goal of this project is:

> *".. to develop new solutions that enable AV streaming services over heterogeneous, mobile and unstable networks which are found for instance in emergency and rescue operations"*

This thesis aims to find alternatives to the streaming of video in resource challenged networks to lower resource usage while making sure that no important information is lost. In this chapter we present background information in Section 1.1, describe the motivation for our work in Section 1.2, present our problem statement in Section 1.3, discuss the methods used in Section 1.4 and finally outline the remainder of this thesis in Section 1.5.

1.1 Background

In emergency situations like natural disasters or large scale accidents, the regular infrastructure for connecting to the Internet might not be available. Modern handheld devices such as mobile phones and PDAs are still able to connect to a network in infrastructureless mode. A network of devices connected in this fashion is referred to as a Mobile Ad-hoc Network (MANET). Such a network could be used to convey information from the incident site to a Command and Control Center (CCC), thus helping personnel at the CCC get a better overview of the emergency situation. A clear overview of the situation is of utmost importance and will enable the CCC personnel to make correct decisions. This thesis focuses on information obtained from video captured by personnel at the incident site carrying head-mounted cameras.

1.2 Motivation

In an Emergency and Rescue (ER) scenario such as the one introduced above, the network available is typically a sparse, disruptive MANET.

Sparse meaning it has a low density of nodes and disruptive meaning links go up and down as nodes move in and out of range of each other. This implies that we might not have a path from the incident site to the CCC up at all times. We might not even have a full path up at all at any one point in time. We therefore utilize the Message Ferrying approach proposed by Zhao et al. [37] for dissemination of data from the source device at the incident site to the receiver at the CCC and vice versa. With this approach we take advantage of non-random mobility in certain nodes (the message ferries) to assure message delivery over time. Emergency rescue personnel will be moving between the incident site and the CCC, taking on the role of message ferries. A typical example of this would be a paramedic in an ambulance.

The inherent properties of a sparse, disruptive MANET presents challenges regarding connectivity and up time of links. Most or all nodes in the MANET will be handheld devices with limited bandwidth and power supply. To be able to make the correct decisions and to do so in a quick and efficient manner, a good overview of the situation is paramount for the personnel at the CCC.

Streaming of video, which has been the subject of research in the DT-Stream project, is a resource costly operation, which will quickly drain the batteries of the mobile devices performing it. We aim to decrease the resource usage to achieve a longer lifetime of the network. The communication between devices is often regarded as the main source of power consumption and should therefore be minimized. We propose applying image analysis techniques to minimize the data that needs to be transferred from the incident site to the CCC while making sure that important information is not lost. In this thesis we present different approaches for applying image analysis with the goal of lowering resource usage while still making sure the important information reaches the CCC.

A substantial amount of research has been done both in the field of video streaming over MANETs and in the different fields of image analysis, but a combination of these fields has to our best knowledge not yet been explored. We hope to make use of the knowledge gained in these areas of research to reach our goal of getting important information to the CCC without putting too much strain on the network.

1.3 Problem Statement

The main problem we are addressing is how we can give the CCC personnel the best possible overview of the emergency situation. This information will be based on footage from on site personnel wearing head-mounted cameras. At the same time we want to preserve the scarce resources of the handheld devices in our disruptive MANET to allow a longer lifetime of the network.

1.4 Methods

In this section we describe the scientific methods utilized during our work with this thesis.

1.4.1 Literature Studies

The basis of any research is the research already conducted by others. For our thesis we have studied literature related to our research, both internal literature from the DT-Stream project and the University of Oslo as well as external literature touching in on the same subjects. As mentioned, a substantial amount of research has been done in the area of Mobile Ad-hoc networks and in the image analysis fields we present, but we find no example of them being combined as we do in this thesis. In our literature studies we have used digital archives of articles and research papers like IEEE Xplore[1], ACM Digital Library[2] and Google Scholar[3].

1.4.2 Design

We propose alternative solutions for conveying information from a video across the network and design a system that can execute a selection of these solutions.

1.4.3 Implementation

We implement the designed system utilizing existing programs and libraries as well as self-implemented programs and applications.

1.4.4 Experiment

Empirical studies is another cornerstone of scientific research. The essence of empirical studies is that observing what happens when performing the same action repeatedly gives a good indication of what will happen the next time that action is performed. In our case, we conduct an experiment with multiple video clips and configurations to measure the effect of different information conveying techniques. We run multiple iterations of each combination of video clip and configuration and look at the average of the results to smooth out peaks caused by coincidences and other outliers. We look at the standard deviation to evaluate if our results can be trusted or if more iterations are needed.

1.4.5 Physical Measurements

In our experiment we physically measure the power consumption of the mobile device while performing specific tasks. Using physical

[1] http://ieeexplore.ieee.org/Xplore/
[2] http://dl.acm.org/
[3] http://scholar.google.com

measurements instead of a power consumption model provides more accurate and realistic results.

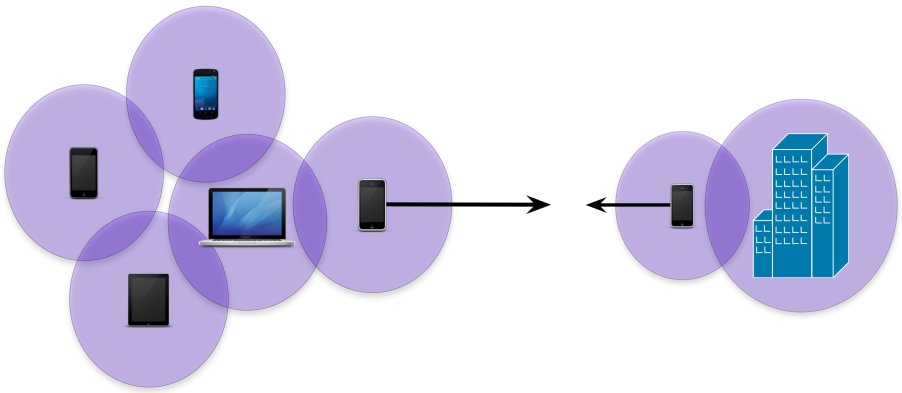
1.5 Outline

The rest of this thesis is structured as follows. In Chapter 2 we present our application scenario, the Emergency and Rescue (ER) scenario, describe its characteristics and discuss its requirements and technical considerations. Chapter 3 describes techniques and methods used for delay tolerant communication and in ad-hoc networks. In Chapter 4 we present different techniques for conveying information based on video across the network. In Chapter 5 we present related work connected to measurement of power consumption and bandwidth usage. Chapter 6 contains the design of an application for the workers at the CCC, consisting of components which utilize these information conveying techniques to present important information from the video captured at the incident site. We outline the requirements and present the system design for the component we implement. Chapter 7 is the Implementation chapter where we describe the implementation of one of the components of the CCC application. We describe the system architecture for the system we implement and look at the flow of data in the system. In Chapter 8 Evaluation we describe the experiment conducted to analyze the resource usage of a state of the art mobile device, while performing required tasks. We describe the setup of the experiment and define its workload, configurations and metrics. We then present and analyze the results of this experiment. Chapter 9 is the conclusion of the thesis. Here, we summarize our work, evaluate our contribution and perform a critical assessment before outlining possible future work.

Chapter 2

The Application Scenario: Emergency and Rescue Operations

In this chapter we describe the application scenario and outline its characteristics in Section 2.1 and requirements in Section 2.2. Our application scenario is the Emergency and Rescue (ER) scenario, a realistic depiction of a real-life emergency situation, where the normal network infrastructure is unavailable due to a natural disaster or other major event.



(a) Incident site (b) CCC

Figure 2.1: Layout of the ER scenario

Figure 2.1 gives an overview of this scenario. The participants in this scenario are the people at the incident site, both rescue workers and civilians, personnel at the CCC and rescue workers moving between the locations, for instance when transporting wounded individuals. The mobile devices of these participants form a sparse Mobile Ad-Hoc Network (MANET) which is used for communication. This is a partitioned network, where the incident site typically contains a relatively large amount of devices, while the area between the incident site and the CCC is sparsely populated. Using delay tolerant networking, video from a camera at the

incident site can be transmitted to the CCC then carries the data as it travels toward the CCC, ultimately delivering the data to the CCC when in range of it.

In their research report, Sanderson et al. [26] presents the characteristics and requirements of a Rescue and Emergency Operation. In the following subsections we will summarize the most important aspects of this type of application scenario.

2.1 Characteristics

A rescue operation is characterized by its type, size and scale, its type being e.g. land, sea or air and its scale being local, national or international. In this thesis we picture a local rescue operation on land, for instance the rescue of victims of an earthquake from collapsed buildings. This section presents the characteristics of our application scenario.

2.1.1 Multiple Organizations

There are typically multiple organizations involved in a rescue operation. The organizations involved could be anything from the police, fire fighters and the Red Cross to rescue dogs and volunteers. In Norway, the police have the main responsibility for the coordination and management at the rescue site. They are also the ones who convene other governmental departments, services or the military when needed. Different departments and organizations have access to different equipment and resources. For instance, the fire brigade have special equipment for rescue operations in tunnels and the coast guard have fast boats for rescue work at sea. It is therefore important that the police summon what is needed for the situation at hand. The police continuously communicate with the county governor to ensure cooperation and mutual understanding. Being responsible for summoning other assets when needed, the police must have as close to full knowledge as possible of what is going on at the incident site.

2.1.2 Hectic Environment

The environment of the rescue operation is often hectic and dynamic. Personnel and resources may come and go and the situation can seem chaotic. In this type of environment with multiple organizations involved, each with their own procedures and guidelines, cooperation is a challeng. However, the incentive for cooperation is strong as every participant share the same goals of rescuing people and limiting damages.

2.1.3 Organizing the Operation

Organizing the emergency and rescue operation can be a challenge. Governmental authorities have guidelines regarding the structure and organization of rescue and emergency operations, including responsibilities and

the chain of reporting. These guidelines lead to a hierarchical structure with different organizations having different responsibilities. The participants in the rescue operation report to their superiors, which in turn report further up in the hierarchy to the On-Site Coordinator which reports to the CCC.

2.2 Requirements and Technical Considerations

In their report [26], Sanderson et al. analyze three possible emergency scenarios, including an earthquake and a railway accident. When looking at the commonalities of these scenarios, the following requirements and technical considerations emerge.

2.2.1 Information sharing: Inter- and Intra-operability

One of the main requirements of the ER scenario is to get information from the incident site to the CCC. As mentioned in the characteristics section there are multiple organizations involved in a rescue operation. Communication both within and between organizations is therefore required and should be as smooth as possible. It would be a benefit if applications could access available information and share it automatically among the participants who need this information, both internally and externally.

2.2.2 Urgency

In an emergency situation, time is of the essence. Any time saved can help save lives and limit damages. It is therefore a requirement that the workers at the CCC get information as fast as possible, enabling them to make educated decisions.

2.2.3 Security

Using a wireless network makes the system vulnerable to attacks. Anyone in the area with a wireless device may listen to the data traffic and create traffic of their own. The system must therefore consider the following security challenges.

- **Authenticity:** Mechanisms are needed to keep unauthorized individuals out of the network.
- **Integrity:** Ensuring that network traffic can not be manipulated, e.g. its content being changed, is necessary.
- **Confidentiality:** Confidential messages must not be available to unauthorized personnel.

2.2.4 Network Considerations and Resource Management

Setting up communication networks in the affected area must happen fast, and no assumptions about the infrastructure in the area can be made. A Mobile Ad-hoc Network (MANET) is a good candidate for this application domain. The movement of the nodes in the network, together with physical obstacles and the layout of the area may lead to frequent disruptions and network partitions, making this a disruptive MANET which requires the use of Delay Tolerant Networking (DTN) techniques.

There is a high degree of heterogeneity between the end-user devices, including everything from laptops to mobile phones. Important resources such as processing power, storage space, bandwidth and battery power must be used efficiently and cautiously. Resource management is needed as there are no guarantees as to what resources are available at a given time, due to the disruptive nature of the MANET and the movement of nodes within the network. Power is perhaps the most important resource of all in this scenario as we require devices to be up to allow for communication. We want the network to stay alive as long as possible and we are therefore required to be cautious with regards to resource usage. Having all devices go down because of battery depletion would prevent any and all communication and should not happen. Wireless communication is an energy draining operation that should be kept to a minimum to allow for the most important information to get through.

2.2.5 Communication issues

There are several aspects to communication that must be considered. You have technological challenges such as different organizations using different frequencies for radio communication or interference from unrelated traffic in the area. Sociological issues must also be taken into consideration. It is, for instance, human nature to want frequent updates as to what is happening in a stressful situation. Another sociological aspect is that different organizations may use different jargon and technical terms. It is important that information is in a format understandable by all receivers. The organization of communication is a third challenge. The system should provide an easy way of cooperating across organizations, should the predefined hierarchies within the organizations prove not to be efficient enough.

Chapter 3

Network Technologies

In this chapter we present network technologies and terms which we refer to later in the thesis. We begin by describing wireless communication in Section 3.1, before presenting mobile ad-hoc networks in Section 3.2. Section 3.3 covers delay tolerant networking while Section 3.4 describes the Dts-Overlay.

3.1 Wireless Communication

The Institute of Electrical and Electronics Engineering (IEEE) have developed standards for communication in wireless local area networks (WLANs) known as the IEEE 802.11 standards. Most of these standards operate on the two open radio bands; the 2.4 GHz industrial, scientific and medical (ISM) band and the 5 GHz band. The most prominent standards in the 802.11 family is the 802.11 a, b, g and n standards. While the 802.11 b and g standards operate on the 2.4 GHz band, the 802.11 a standard operates on the 5 GHz band. 802.11 n, however, can operate on both bands. In theory, the b standard can provide up to 11 Mbps data rate, the a and g standards can provide up to 54 Mbps and the n standard can provide up to 100 Mbps.

Wi-Fi capable devices can communicate with each other by using radio waves on the aforementioned frequencies. The communication range of the devices depend on a series of factors, such as the antenna, the environment (inside or outside, with or without obstacles) and what frequency band is used. With optimal conditions the range can be up to several hundred meters.

There are two main types of networks that Wi-Fi capable devices can take part in; infrastructure networks and ad-hoc networks. Infrastructure networks are the networks we are used to at home and at the office, which have one or more access points (APs). In such a network, devices connect to an AP, which controls communication both locally between the devices and remotely to other networks such as the Internet. In an ad-hoc network, on the other hand, the devices communicate directly with each other. The AP is the main difference between an infrastructure network and an ad-hoc network. While the AP routes all messages in an infrastructure network,

each device needs to take on the role as router in an ad-hoc network.

In today's world there is a plethora of devices capable of Wi-Fi communication. From computers, mobile phones, PDAs and sensors to refrigerators and cars.

3.2 Mobile Ad-hoc Networks (MANETs)

In our application scenario the regular Internet is not available, so we must utilize another way of networking to transmit data from the incident site to the CCC. As mentioned in Chapter 1 Introduction, modern mobile devices can create their own network without the need of existing infrastructure. Such a network is called a Mobile Ad-hoc Network (MANET).

> *"A MANET, is a system of wireless mobile nodes that can freely and dynamically self-organize into arbitrary and temporary network topologies, allowing people and devices to seamlessly internetwork in areas without any pre-existing communication infrastructure."* [8]

A MANET is typically a heterogeneous network, meaning it consists of multiple types of devices. The primary challenge in building a MANET is making each device able to continuously maintain the information required to properly route traffic. MANETs may operate by themselves or may be connected to the larger Internet. In addition to being able to send or receive data, each device in a MANET should be able to take on the role as forwarding router for passing on packets from other devices. A MANET is dynamic by nature, meaning its topology changes as devices move in and out of range of eachother, breaking and establishing links as they go.

In our application scenario, as described in Chapter 2, the MANET will typically have a high density of devices at certain locations (the incident site and the area surrounding the CCC) and a substantially lower density in between them. This implies that the network will be fractioned and that there most likely will not be a complete path from the source at the incident site to the destination at the CCC at any one time.

One of the most important aspects of the communication process is the design of the routing protocols used to establish and maintain multi-hop routes to allow the communication of data between nodes. Routing is a challenging task due to the salient characteristics of the MANET such as dynamic topologies, bandwidth constrains, variable capacity links, energy limitations and physical security issues. In addition, MANETs come in different shapes and sizes. From sparse to dense and from small to large. This diversity in MANETs has inspired the development of an array of different routing protocols, often specifically built to meet the needs of the author within a certain scenario. Some of the more general and most widely used protocols are Optimized Link State Routing (OLSR) and Ad-hoc On-demand Distance Vector routing (AODV). In the following sections we present these routing protocols.

3.2.1 Optimized Link State Routing (OLSR)

OLSR [9] is a proactive routing protocol, meaning that routes to all destinations in the network are known and maintained before they are used. This implies that there is no delay caused by route discovery. OLSR uses Hello messages and Topology Control messages to discover and distribute link state. The protocol is optimized for MANETs and works best in large, dense networks. When routing, each node makes use of the topology information to select the next shortest hop. OLSR limits the amount of flooding when distributing link state information by using Multipoint Relays (MPRs).

Characteristics of Link State Routing

OLSR has certain characteristics which it shares with other link state routing protocols. Link state routing protocols are proactive which means they keep information about the topology and state of the network so that routing information is already available when communication is initiated. They aim to give all nodes overview over the state of all links in the network and achieves this by propagating local information throughout the network. In link state routing each node periodically broadcasts link status. Each node is also responsible of rebroadcasting link state information received from its neighbors, and to keep track of the information received from other nodes. Under transmission the link state information gathered is used to determine the next hop along the path to the destination.

Multipoint Relays

OLSR utilizes multipoint relays (MPRs) to optimize the flooding of link state. The main idea is to minimize the amount of broadcasting nodes, by intelligently selecting the minimal set of nodes that cover all other nodes in the network. When selecting which nodes to use as MPRs there are two main rules:

- Every 2-hop neighbor must be covered by at least one MPR.
- The number of MPRs should be minimized.

The following is an algorithm for selecting MPRs while adhering to these rules. For each node:

1. Start with an empty MPR set.
2. For all 2-hop neighbors that only have one direct neighbor: add that neighbor to the set of MPRs.
3. While any 2-hop neighbors are not yet covered: add the node that covers the maximum amount of 2-hop neighbors to the set of MPRs .

When the minimum amount of MPRs that cover all two hop neighbors have been selected, the cost of flooding will be substantially decreased compared to a complete flooding approach. Additionally, only MPRs need to forward control messages and generate link state information because they cover all links in the network.

3.2.2 Ad-hoc On-demand Distance Vector Routing (AODV)

AODV [22] is a reactive routing protocol intended for use in a MANET. A reactive routing protocol is a protocol that does not keep a record of routes that are not currently in use for communication. Instead it is able to quickly discover routes on demand. Not having to keep a routing table makes the system more lightweight, but the route discovery can cause latency. Being a reactive protocol, AODV is able to quickly react to link failures or changes in topology. A typical target network for AODV is a network where routes change often, for instance due to mobility or link failure. In such a network the cost of maintaining a list of routes in a proactive manner would outweigh the benefits.

Route Discovery

Since AODV does not keep track of routes that are not currently in use, routes are discovered on-demand. To discover a route, a Route Request (RREQ) is broadcast throughout the network. Each node in the network must broadcast this RREQ only once. Each intermediate node includes the id of the node they got the RREQ from when forwarding it. When the RREQ reaches the destination node, this informaton is used to form a reverse path back to the soruce node. A Route Reply (RREP) is then sent along this reverse path. When the RREP reaches the source node, the route has been established.

Route Maintenance

All intermediate nodes maintain reverse paths for a RREQ, but only some of them, namely the ones that are part of the shortest route, will get the RREP to send along the reverse path. A timeout is used so that unused reverse paths are purged from the routing table. However, the timeout obviously should be long enough to allow the RREP to come back along the shortest reverse path. Forward routes can also be purged. The purging of a forwarding route occurs when it has been inactive for too long. If no data is sent along a forward route for a set amount of time, the entry for that route is removed from the routing table.

Route Error

When a node can not forward a packet along the link defined in the discovered route it generates a Route Error (RERR) message. This RERR message is transmitted along the reverse path back to the source node.

Upon receipt of the RERR message, the source node initializes a new route discovery.

3.3 Delay Tolerant Networking (DTN)

Standard communication protocols and techniques often make assumptions that are not feasible when the target network is a disruptive MANET, such as the assumption that we have a complete path from the source to the destination or that a packet will arrive within a certain time limit. To be able to communicate in a fractioned and dynamical network, such as a disruptive MANET, we therefore need to utilize delay tolerant communication techniques.

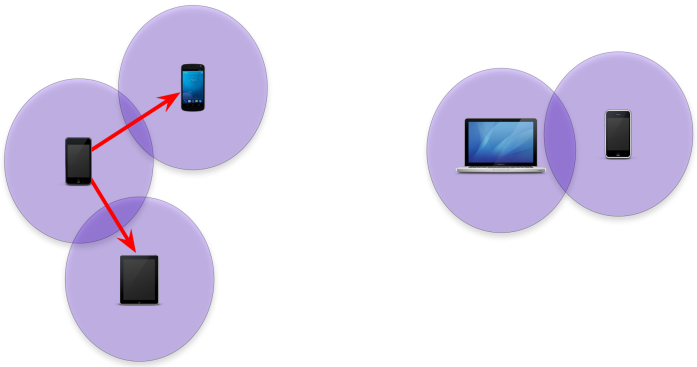
Two prominent delay tolerant communication techniques are Epidemic Routing and Message Ferrying. One main difference between the two is how messages are forwarded. With epidemic routing, carriers forward messages to every node they come in contact with that does not have them already. Message ferries instead rely on a routing table and only forwards the packet along the way to the destination. Additionally, the movement pattern of message ferries is often known in advance, while epidemic routing can work with random node movement.

3.3.1 Epidemic Routing

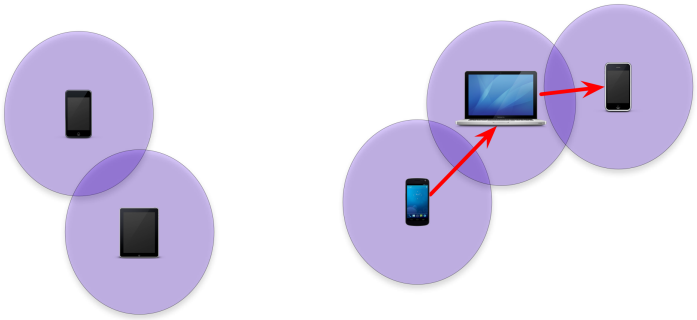
Epidemic Routing, introduced by Vahdat and Becker [30] is a method where random pairwise exchange of messages between mobile devices lead to eventual message delivery. The first step is distributing the message to nodes within the partition of the network the source device is a part of. Due to node mobility some of the nodes will come in contact with other nodes or network partitions and thus the message is spread to other portions of the network. This method gives a high probability of the messages eventually reaching their destination. The same principal could be applied in out application scenario using data packets instead of messages. Figure 3.1 shows an example of how epidemic routing works.

When using the epidemic routing protocol, each node contains a buffer of messages, both originating from the node itself and from other nodes in the network. Every message has a unique identifier which is the key for that particular message in a hash table. Each node also keeps a summary vector of what messages it currently holds. When two nodes, A and B, come within communication range of each other, they exchange summary vectors. Both nodes checks which of the other node's messages it is missing and requests these messages. In the final step, node A transmits the messages that node B requested to node B and vice versa.

Epidemic routing has its disadvantages. Among them is the potentially high failure rate (we can not be sure that the message will reach the destination) and the extra resource consumption for each node, caused by the duplication of messages. Since one of our main goals is to reduce



(a) The source disseminates the message within its own network partition



(b) Due to mobility, one of the nodes holding the message comes in contact with the network partition containing the destination and the message arrives at the destination

Figure 3.1: Epidemic routing

resource consumption, epidemic routing is not the ideal solution for our application scenario.

3.3.2 Message Ferrying

Message Ferrying [37] is a technique for information dissemination in a partitioned network, taking advantage of predefined or predictable movement of certain nodes in the network. Forwarding through a message ferry consists of three main steps:

1. **Store:** The message ferry discovers that the next hop is not within reach. It stores all packets for future transmission.
2. **Carry:** The message ferry carries the packets closer to the destination.
3. **Forward:** When in range, the message ferry forwards all stored packets to the next hop.

Message ferrying can be divided into two types: Node-initiated message ferrying (NIMF) and ferry-initiated message ferrying (FIMF). In

NIMF, nodes move within range of the route of the message ferry when they have something to transmit, while in FIMF the ferry moves to meet the nodes. FIMF is the delay tolerant method of choice for our application scenario. We have chosen this method because we have certain nodes with predictable movement between the two main network partitions. These message ferry candidates are the devices of rescue workers moving back and forth between the incident site and the CCC, for instance the devices of ambulance personnel transporting wounded individuals to a hospital. Figure 2.1 showing the layout of the ER scenario also demonstrates the principle of message ferrying. Here, the devices with arrows are the message ferries moving back and forth between the incident site and the CCC.

3.4 Dts-Overlay

In order to transmit video and images over our disruptive MANET, we utilize the Delay Tolerant Stream Overlay (Dts-Overlay) developed by Lindeberg et al. [16]. The main task of the Dts-Overlay is to decide where to forward packets and to store packets when they cannot be forwarded or when forwarding them is not meaningful. Figure 3.2 shows the protocol stack for a client (C), server (S) and intermediate node (I) when using the Dts-Overlay. Worth to note is that OLSR is used as the routing protocol and UDP is used as the transport layer protocol.

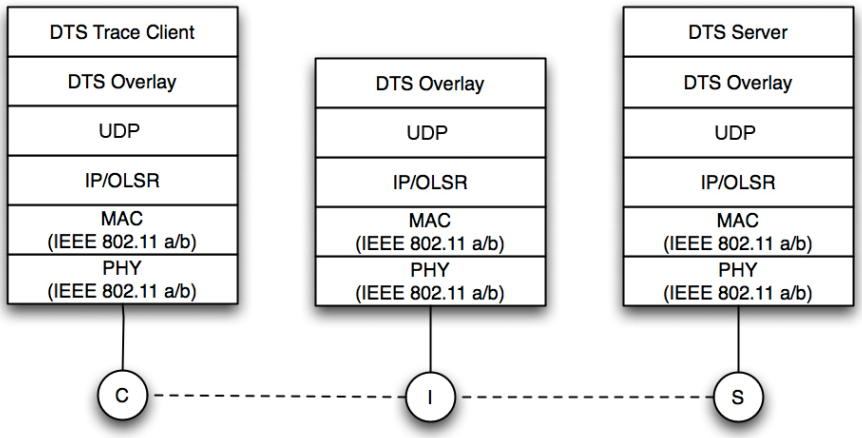


Figure 3.2: Protocol stack when using Dts-Overlay

Dts-Overlay forwards packets hop-by-hop with the goal of transporting the packet as close to the destination as possible. As mentioned in the MANET section, the links between devices are volatile due to mobility. Precautions must be taken to avoid packet loss due to low link quality. The Dts-Overlay has mechanisms to deal with this issue. Packet forwarding in the Dts-Overlay is handled as follows:

1. **If OLSR can not find a route to the destination, we attempt to use recent routes from the network topology history.** The network topology history is kept by Dts-Overlay.

2. **If the next-hop of one of these routes is within range, we forward the packet to that node.** This because the next-hop of a recent route is likely to be closer to the destination than the current node.

3. **If there are no recent routes, we store the packet in a buffer in the Dts-Overlay.** This avoids unneccessary transmission and thus lowers packet loss.

4. **If we find a next-hop, we check the MAC layer retransmission queue.** If this queue is filling, the link quality is likely poor and we temporary suspend transmission.

As mentioned earlier, we utilize message ferries to transport data from the incident site to the CCC. For efficient identification of these message ferries, we assume they have fixed IP adresses which are known to the other nodes in the network. When a path to the destination is not found, but a route to a message ferry exists, we forward packets along that route. To avoid loops, we do not allow transmission between message ferries.

Chapter 4

Techniques for Conveying Visual Information

There are numerous ways of conveying information from video. In this chapter we go through a handful of them. The techniques we look into are transmission of uncompressed video, transmission of compressed video, episode detection and key frame selection, face detection and image stitching. We have chosen these techniques as we deem them suited for our application scenario. In the following sections we describe each technique further and discuss their advantages and disadvantages.

4.1 Transmission of Video

The first that comes to mind when considering how to convey information from video across a network is to transmit the video itself. As mentioned in Chapter 1 Introduction, streaming of video is the subject of research for the DT-Stream project and has been the focus of papers by Lindeberg et al. among others [10][15][16][17][24]. In the experiment we conduct we use transmission of the video as it is when received from the camera as a baseline to compare our other approaches with.

Advantages of Transmitting Uncompressed Video:

- **No information lost.** Since the whole video is transmitted, the personnel at the CCC sees all that the rescue worker sees.
- **No loss of video quality.** The video received at the CCC is of the same quality as the video captured at the incident site.
- **No additional processing.** Preprocessing at the source device is not needed when the video is transmitted as it is.

Disadvantages:

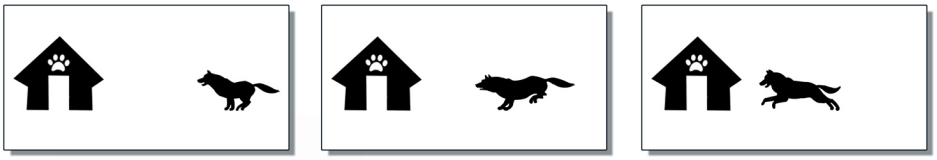
- **Large file size.** The large file size leads to a high amount of transmission related resource usage.

- **Time wasted.** Personnel at the CCC spends valuable time looking through videos to uncover important information.

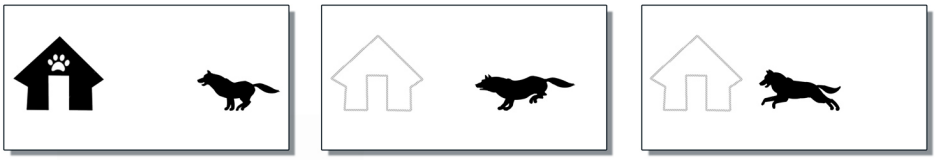
4.2 Transmission of Compressed Video

We argue that the video file as it is when received from the camera is unnecessarily large to be suitable for transmission in our resource limited network. We therefore utilize video compression to lower the file size of the video. In our experiment we compress the video received from the camera using the MPEG compression technique and subsequently transmit the compressed video.

A video sequence is a series of images taken at a fast rate. When from the same scene, there is typically not a large difference between one of these images and the next. This is taken advantage of by video compression techniques used in the MPEG (Motion Picture Experts Group) standard [21] and other compression standards. The technique is known as *difference coding*. A compression technique which bases itself on information from other images is called an *interframe* technique. When a scene change occurs, the first image from the new scene generally does not resemble the last image from the previous scene. Therefore another compression technique is needed for the first image of each scene. To compress a single image we take advantage of similarities between pixels in regions of the image. A compression technique which is based on information from a single image is called an *intraframe* technique.



(a) Original images



(b) Difference coded frames

Figure 4.1: Difference coding

Figure 4.1 shows the principle of difference coding. The first frame is coded with an intraframe technique while the other two are coded with an interframe technique taking advantage of the similarities between the frames. Since the dog is moving, it is included in the two later frames, while the stationary dog house is not repeated.

4.2.1 MPEG Compression

An MPEG compressed file consists of a series of MPEG group of pictures (GOPs). Such a GOP, as seen in Figure 4.2 normally consists of three types of frames: I-frames, P-frames and B-frames.

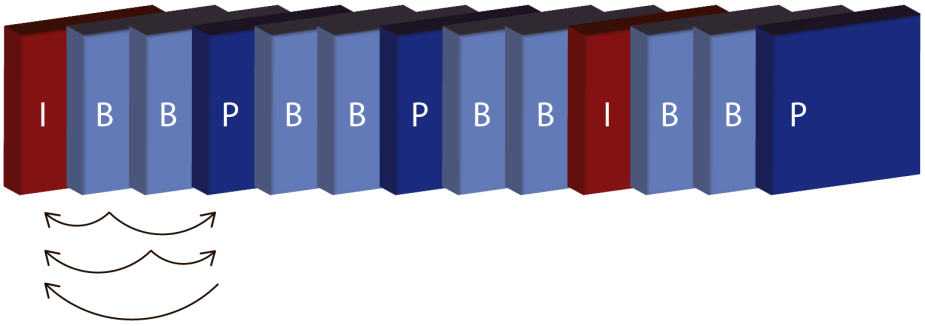


Figure 4.2: An MPEG Group of Pictures (GOP)

I-frames (intra-coded frames), often referred to as key frames, uses an intraframe compression technique as described above. This means that its content is examined independently. Preceding or succeeding frames are not taken into account. This is essentially the same as image compression such as when using the JPEG compression standard. I-frames have the smallest degree of file size reduction of the three frame types.

P-frames (predictive-coded frames), often called delta frames, are compressed using information from preceding I-frames or P-frames. A P-frame is typically a frame where parts of the content or the color has changed from the preceding I-frame or P-frame. Since it only contains the difference, it does not make sense to display a P-frame on its own. The I-frame is needed to fill in the original content of the video. In Figure 4.2 we see that the first P-frame is based on information from the preceding I-frame.

B-frames (bidirectionally predictive-coded frames), also a subgroup of the expression delta frames, may use information from both preceding and succeeding I-frames or P-frames in the group of pictures. Like a P-frame it does not make sense to display a B-frame on its own. It requires the information from the I-frames and P-frames it is based on to be displayed. In Figure 4.2 we see that the two first B-frames are based on information from the preceding I-frame and the following P-frame.

4.2.2 Advantages and Disadvantages of Video Compression

In this section we list certain advantages and disadvantages of compressing video and transmitting the compressed video.

Advantages:

- **No lost information:** Since the whole video is transmitted, the personnel at the CCC sees all that the rescue worker sees.
- **Less data:** The compressed video is substantially smaller than the uncompressed video, meaning less data needs to be transferred.

Disadvantages:

- **Large file size:** Compressed video files are still large and transmitting them consumes a substantial amount of resources.
- **Power consumption from processing:** The actual compression operation consumes power from the battery of the source device.
- **Time wasted:** Personnel at the CCC spends valuable time looking through videos to uncover important information.

4.3 Episode Detection and Key Frame Selection

A common way to summarize video is to first divide it into meaningful segments, then select a frame from within each segment to represent that segment. In the normal case, when working with edited video, we can divide the video into shots using shot transitions such as cuts, fades or dissolves to separate them. These transitions could occur when switching focus from one person to another in an interview situation or when moving from one scene to another in a motion picture. Selecting a frame from within a shot, a so called key frame, is a common way of representing or summarizing the shot. In our application scenario, the video is captured from the head-mounted camera of a rescue worker. This video is unedited and originates from one continuous roll of the camera and therefore does not have the same kind of transitions as an edited video would.

4.3.1 Key Frame Selection Strategies

There are numerous key frame selection strategies. A basic approach to key frame selection in this scenario would be selecting a new key frame at a set time interval, but this would not be ideal. If the interval is too short, the CCC would receive frames with little or no new information and if the interval is too long, the CCC could miss out on important frames. More advanced selection strategies take metrics into account, e.g. basing the key frame selection on a movement metric as done by Wolf [32]. A preferred approach for our continuous roll from a head-mounted camera would be to split the video into meaningful episodes and choose a key frame from each episode. This is the approach taken by Chauhan et. al in their article on *Episode detection in videos captured using a head-mounted camera* [6].

4.3.2 Episode Detection

We define an episode as a part of the video where the rescue worker with the camera is focusing on one thing. The transitions between such episodes are not as clear as those between traditional shots, such as cuts and fades. The video image instead gradually changes as a result of camera movement or moving objects in the image space. To detect a cut in an edited video, one could look at the difference of certain features between one frame and the previous. In our case it is not enough to look at two consecutive frames. Instead we should compare each frame with the last selected key frame. We could say that we have a new episode when there is a certain difference between some features of the current frame compared to the same features of the last key frame, meaning that the camera has moved a certain amount or there has been significant movement in the image space. Chauhan et. al [6] uses another approach to detect episode transitions:

> *"Our research is based on the assumption that head movements have distinguishable patterns during an episode occurrence and these patterns can be exploited to differentiate between an episode and a non-episode."*

No matter what the preferred method for episode detection is, dividing the video into meaningful episodes helps remove unimportant parts of the video and makes it easier for the personnel at the CCC to find the most important information. When episodes have been detected and key frames selected, the key frames can be transmitted to the CCC and the personnel at the CCC, based on these images, could choose which episode(s) they want to receive first.

4.3.3 Advantages and Disadvantages of Episode Detection and Key Frame Selection:

In this section we list a number of advantages and disadvantages related to using episode detection and key frame selection as a technique for conveying visual information.

Advantages:

- **Less data:** Smaller parts of the video are transmitted meaning a smaller amount of data in the network.
- **Episode requests:** The CCC can request certain episodes in the video, based on the key frames. This can help make important information arrive faster.

Disadvantages:

- **Power consumption from processing:** Detecting episodes and selecting good key frames require processing and consume resources at the source device.

- **Overlooking important information:** Vital information might be missed by the personnel at the CCC if the key frames do not properly reflect the contents of an episode.

4.4 Image Stitching

Image stitching [28] is the act of combining multiple overlapping images into a panoramic image. Figure 4.3 shows an example of image stitching performed on two overlapping images to form a panoramic image, while Figure 4.4 shows a panorama created by stitching four partially overlapping images.

Figure 4.3: An example of image stitching

Image stitching is one of the oldest and most widely used techniques in the field of computer vision. Modern digital cameras often come with integrated image stitching algorithms allowing the user to easily create panoramic images. In the CCC application we use image stitching as a way of summarizing a swiping motion in a video in order to give the personnel an overview of the incident site without consuming too much of the limited resources of our network. Image stitching algorithms can be divided into to two main categories. While one works by minimizing pixel-to-pixel dissimilarities, the other is based on trying to match features in the images that are to be stitched [3][2]. For our image stitching application, we have chosen an algorithm from the latter category.

4.4.1 Single Viewpoint

For a successful panoramic image, all images must be taken from the same viewpoint. Failing to do this can cause parallax. Parallax is the effect that

Figure 4.4: A panorama made from four images

makes two objects appear to change position relative to each other when viewed from different viewpoints. The relative movement of objects caused by parallax can prevent images from being properly stitched, thus parallax should be avoided if possible. In Figure 4.4 we see examples of parallax in the diamonds along the edges of the pool table.

4.4.2 Camera Settings

If possible, the automatic exposure adjustments common in most modern cameras should be switched off to make sure that the whole video is taken with the same exposure settings. Allowing the automatic adjustments can lead to differences in brightness in different frames of the video making stitching more complicated or resulting in an inconsistent panorama. The stitching algorithm does however compensate for differences in brightness between individual images.

4.4.3 Selecting Frames

It is important to select frames suitable for image stitching as they need to have a certain amount of overlap to be able to be stitched. One approach would be selecting frames at a fixed rate, e.g. every 20th frame or every second. Another would be selecting the frames based on certain criteria. When using the fixed rate approach, the rate of frames to stitch depends on the speed of the swipe. If the swipe is fast, we have to extract frames more often than if the swipe is slow. However, a swipe does not necessarily

have a constant speed, which is one of the limitations of this approach. If two frames are chosen, and the camera movement between the two is minimal, unnecessary processing power is used on stitching these frames. In this case it would be better to stitch one of them with a later frame. To avoid this, we could operate with minimum and maximum thresholds for the amount of overlap of images that are to be stitched. The lower threshold would be the minimum percentage of overlap that is needed for the stitching to succeed, while the maximum percentage of overlap is used as the upper threshold to avoid unnecessary stitching.

4.4.4 Where to Perform the Image Stitching

An important issue is where the actual image stitching operation should take place. Being a processor heavy task, performing all stitching at the source device would quickly drain its resources and would not be an optimal solution. A more resource friendly solution might be distributing the image stitching task to multiple devices at the incident site, but this would come at the cost of additional wireless communication which is another costly operation.

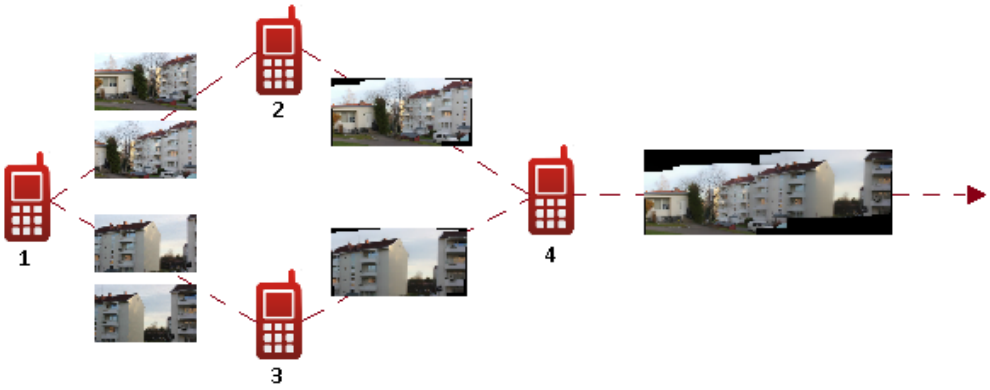


Figure 4.5: Distribution of Image Stitching

In Figure 4.5 device 1 is the source device. Four images have been selected for stitching. To distribute the image stitching task, two of the images are sent to device 2 and two are sent to device 3. Each of these devices then perform stitching on the images received and transmits the resulting image to device 4. Upon receiving these images, device 4 stitches them together and sends the final panoramic image on its path towards the CCC. The idea behind this approach is to distribute the power consumption more evenly among the participating devices to prolong the lifetime of the network

A third approach would be sending all the frames selected for stitching to the CCC and let the computer(s) at the CCC perform the actual stitching. We assume that these computers have virtually unlimited power and superior processing capability compared to that of the mobile devices in the field, making the stitching faster and prolonging the lifetime of the network.

4.4.5 Image file formats

Images are compressed using either lossless or lossy compression algorithms. Lossless algorithms reduce the file size without reducing the quality of the image, and is therefore preferred when quality is more important than file size. An example of an image format using lossless compression is PNG (Portable Network Graphics). Lossy algorithms reduce the file size much more at the cost of also reducing the quality of the image. JPEG (Joint Photographic Experts Group) images are compressed using a lossy algorithm. Since our goal is to lower network load it would not make sense to save the extracted frames in a format that makes their combined file size larger than that of the compressed video. Optimal image quality is not a requirement for this application scenario. Thus, we have chosen to save the key frames as .jpg images using the JPEG compression method.

4.4.6 Image Stitching Algorithm

For image stitching we adhere to the paper by Brown and Lowe [3]. The algorithm presented in their paper consists of the following steps:

1. Feature Matching
2. Image Matching
3. Image Rendering
    (a) Bundle Adjustment
    (b) Gain Compensation
    (c) Multi-band Blending

We elaborate on these steps and how they are executed in the following sections.

4.4.7 Feature Matching with SIFT

The first step of the panoramic recognition algorithm is extracting and then matching Scale-Invariant Feature Transform (SIFT) features from all of the images that are to be stitched. SIFT is an algorithm to detect and describe features in an image, published by David Lowe in 1999 [18][19]. SIFT features are keypoints in image regions which are extracted by the SIFT detector. Their descriptors are then computed by the SIFT descriptor. The four main stages used to generate a set of SIFT features, as outlined by Lowe [19] are:

1. **Scale-space extrema detection:** All scales and image locations are searched to identify potential interest points that are invariant to scale and orientation.

2. **Keypoint localization:** A detailed model is fit at each of the potential interest points to determine location and scale. Keypoints are then selected based on their stability.

3. **Orientation assignment:** Each keypoint is assigned one or more orientations. Future operations will be performed on data that has been transformed relative to this orientation as well as the scale and location of the keypoint. This makes the technique invariant to these transformations.

4. **Keypoint descriptor:** The local gradients around each keypoint is measured at the selected scale. They are then transformed into a representation that allows local shape distortion and illumination change.

SIFT Detector

A SIFT keypoint is a circular image region with an orientation. Figure 4.6 shows an example of such a keypoint. The geometric frame describing it has 4 parameters: the x and y coordinates of the center of the circle, its radius (the scale) and its orientation (the angle).

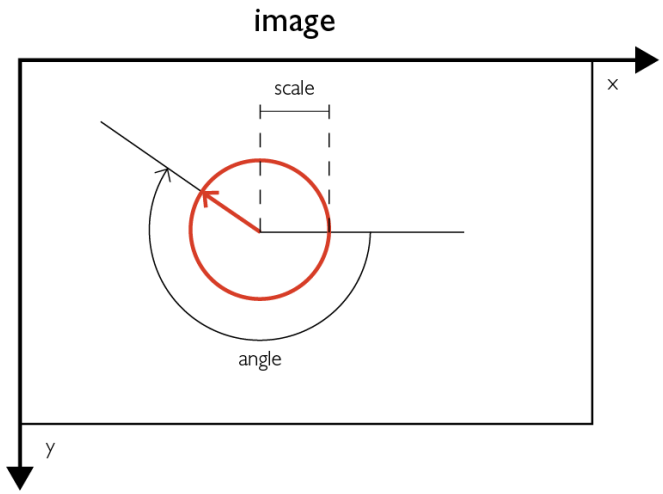


Figure 4.6: A SIFT keypoint

Keypoints are blobs detected by the SIFT detector. It searches for these blobs at different scales and positions, making the detector invariant to translation, rotation and rescaling of the image. After detecting a number of keypoints, those that are likely to be unstable are filtered out. A keypoint could be unstable if it is near the edge of the image or if it is within a low contrast image structure. Thus, filtering of keypoints depends on two thresholds: The *Peak* threshold which is the minimum amount of contrast a keypoint can have and the *Edge* threshold which decides how far from the image edge a point has to be to qualify as a keypoint.

SIFT Descriptor

A SIFT descriptor is a 3D spatial histogram of the image gradients of a keypoint, as seen in Figure 4.7 The gradient at each pixel is formed by the location of the pixel and the orientation of the gradient. Samples are weighed by the gradient norm and accumulated into a 3D histogram. It is this histogram which forms the SIFT descriptor of the image region. A Gaussian weighting function is then applied so that gradients closer to the keypoint center get a higher importance than gradients further away.

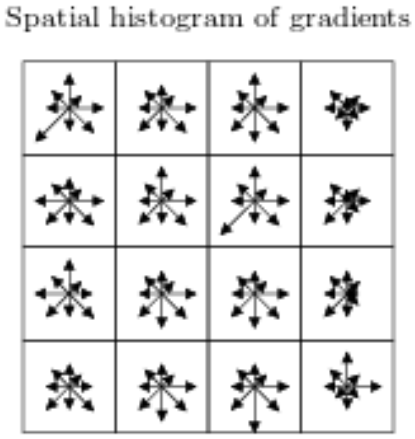


Figure 4.7: A SIFT descriptor [31]

Figure 4.8: A SIFT keypoint with a SIFT descriptor
Photograph: Ole Christian Lingjaerde/Miriam Ragle Aure

Figure 4.8 shows a SIFT keypoint with a SIFT descriptor, while Figure 4.9 shows two images taken by the same camera and the matches between keypoints in the two images.

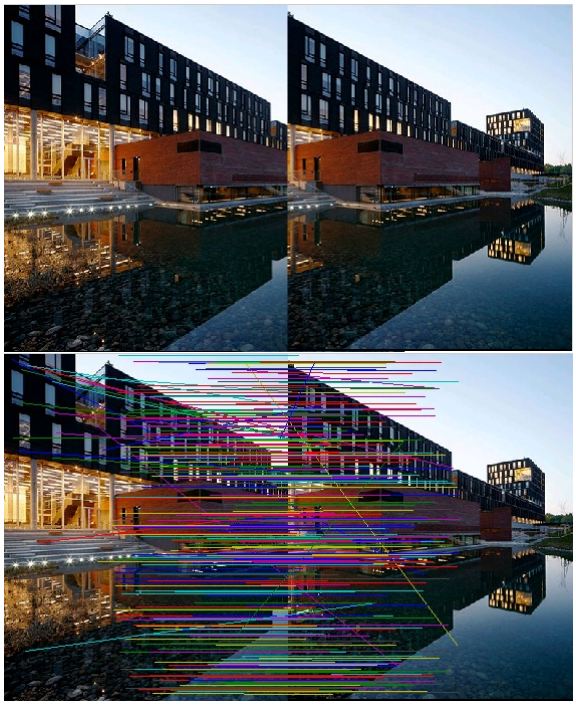


Figure 4.9: Top: Two images from the same scene
Bottom: Matches between keypoints in the two images
Photograph: Ivan Brodey

4.4.8 Image Matching with RANSAC

Once the features are found, the next objective is to identify all overlapping images. Sets of such overlapping images are later used to create panoramas. From the feature matching step, images with a large number of matching features have been identified. In the image matching step, a constant number of images with the greatest number of matches to the current image are considered potential image matches. During the image matching step, RANSAC (RANdom SAmple Consensus) is first used to select a set of inliers that are compatible with a homography between the images. Subsequently, a probabilistic model is used for verification of the match.

The RANSAC algorithm is a robust estimation procedure that estimates image transformation based on a minimal set of random samples. The best solution is then found by examining the estimate's consensus with the data. RANSAC was first introduced by Fischler and Bolles in 1981 [11] as a method for estimating the parameters of a certain model with a set of data containing a substantial amount of outliers. A piece of data is considered an outlier if it does not fit the true model instantiated by the true set of parameters within some error threshold. The two main steps of the RANSAC algorithm are:

- **Step 1: Hypothesize**: Select minimal sample sets randomly from the

data set and use them to determine the model parameters. A minimal sample set is the minimum amount of samples needed to determine the model parameters.

- **Step 2: Test**: Check which elements in the entire data set that coincides with the model using the parameters found in the first step. The set of these elements is called a consensus set.

Consensus sets are ranked by how many elements they contain. The more elements, the higher the rank. The RANSAC algorithm terminates when the chance of finding a better ranked consensus set drops below a certain threshold. In Figure 4.9, most of the matches are horizontal lines of the same length, while you can see certain diagonal lines which are outliers. Because of the high percentage of corresponding matches in this example, the RANSAC algorithm has a high probability of quickly finding the correct consensus set.

4.4.9 Bundle Adjustment

Two images of the same planar surface is related by a homography. Joining such pairwise homographies may cause accumulated errors. To avoid this, the image stitching algorithm uses bundle adjustment. Instead of stitching images and computing their homographies pairwise, images are added to the bundle adjuster one by one in order of how well they match. The added image is then given the same rotation and focal length as the image which it best matches.

4.4.10 Image Rendering

When images have been matched and stitched together, there is still some processing to be done to get a good result. When capturing the video the camera is unlikely to be perfectly level and untilted. This can have the effect that the panoramic image appears wavy. This is corrected by performing *automatic straightening* which is the first step in the image rendering pipeline. The second step is *gain compensation*. Gain compensation is used to avoid large differences in brightness between the images that make up the panorama, caused by automatic adjustments made by the camera. After gain compensation the whole panorama should have the same brightness, but some image edges might still be visible due to effects such as vignetting (intensity decreases close to the edge of the image) and parallax. To smooth out these edges, *multi-band blending* is applied.

4.4.11 Advantages and Disadvantages of Image Stitching

In this section we look at the advantages and disadvantages of using image stitching to summarize parts of a video and transmit the resulting images instead of the entire video.

Advantages:

- **Less data:** A smaller amount of data is transmitted as the stitched image is substantially smaller than both the uncompressed video and the compressed video.
- **Better overview:** A panorama image obtained by using image stitching, arguably gives a better overview of the incident site than a video does.

Disadvantages:

- **Power consumption from processing:** Processing time and resources used for extracting frames and stitching them quickly drains the battery of the device
- **Only applicable to parts of the video:** Stitching images from a video without any notable camera movement has no purpose. Using image stitching to summarize video segments is thus only applicable to parts of the video with substantial camera movement, such as a swiping motion.

The location of the image stitching operation can impact the performance and power consumption. Below are some advantages of performing the image stitching operation at the CCC as opposed to the source device.

Advantages of Image Stitching at the CCC:

- **Near unlimited resources:** Resource usage is not a concern at the CCC, since the computers there do not have the same resource restrictions as the mobile devices at the incident site.
- **Speed:** The computers at the CCC have a higher processing power than the source device making the stitching operation faster.

4.5 Face Detection

During an emergency situation it is likely that the personnel at the CCC would want to get an overview of the number of people at the incident site. To give an estimate of this we utilize face detection.

4.5.1 Idea

An important bit of information in a video from an emergency situation is the number of people in the shot. For instance, when filming a burning building we would want to see how many people are hanging out of the windows. This has motivated us to use face detection as a technique to extract important information from the video without having to transmit and look through the whole video. Face detection could also be used in conjunction with the Episode Detection and Key Frame Selection method. In this case we could detect the number of faces in each frame and define an

episode transition as the event where the number of faces change from one frame to the next. As a key frame we could then select the first frame with a new number of faces. Using the change in the number of faces as a sign of something important happening is similar to how Chauhan et al. [6] uses the appearance of the person wearing the camera's hands as an important event.

4.5.2 Background

Face detection is a form of object recognition. In short, a face detector locates faces in images by looking for features that are typical for faces. OpenCV comes with a face detection program that utilizes Haar-like features which tell us something about oriented contrasts between regions in the image. A set of such features can be used to represent contrasts typical of a human face and their spatial relationships. Making an object detection program starts with training a classifier, in this case a cascade of boosted classifiers working with Haar-like features. A cascade is a combination of several simpler classifiers that are applied subsequently until the candidate is rejected or has passed all the stages. The classifier is trained with hundreds of sample images containing the object we want to detect, called positive examples, and arbitrary images without occurrences of the object, called negative examples. All the positive and negative examples are scaled to the same size. After training the classifier it can be used for detecting objects of the same size as in the training, outputting a 1 if the region it is applied to is likely to contain the object and a 0 if it is not. Moving the detection window across a whole image and checking every location makes it possible to detect all the occurrences of the object in the image. By scaling the classifier and repeating the image scan procedure, objects of different sizes can be found.

Included in the OpenCV library are classifiers for frontal faces, profiles, eyes, etc., some of which we utilize in our face detection. Similarly you could train a classifier to detect any kind of object. In the ER scenario, it could be useful to detect such things as fire or objects with certain colors, depending on the emergency at hand.

4.5.3 Challenges

Faces in images come in many different shapes, colors and angles. A face seen from the front and a face in profile are two different looking objects and are thus detected using two different classifiers. We may run the face detection algorithm multiple times with different classifiers to detect more faces, but this of course comes at the cost of longer processing time and higher resource usage.

False positives is a big issue when it comes to image detection, as many different objects or combination of objects may look like a face. In addition there is of course no way for the face detection program to distinguish between real faces and images of faces. The amount of false positives might mean that face detection is not reliable enough for use in

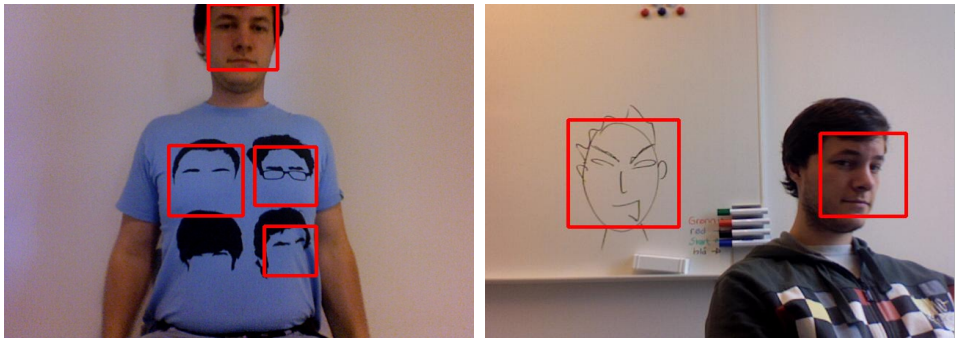


Figure 4.10: False positives when performing face detection

our application. Examples of false positives can be seen in Figure 4.10. Our goal is not to make the ultimate face detection program with the lowest degree of error, but to show that face detection can be used as a tool for extracting important information from videos and for defining episode transitions. For the latter, false positives may result in false episode transitions being detected, leading to unnecessary key frames being extracted and transmitted and thus a higher resource consumption.

4.5.4 OpenCV Face Detection

OpenCV comes with a face detection program which detects faces based on a Haar classifier. It takes images, video or the direct stream from the computer's camera as input, and draws rectangles around detected faces. When using face detection to define episode transitions we alter the program to save a new image, a key frame, when the number of faces changes from the previous frame in the video. When using face detection to detect the total amount of people at the scene, we can run the face detection program with the stitched image as input.

4.5.5 Advantages and Disadvantages of Face Detection

In this section we present advantages and disadvantages of using face detection as a way to extract valuable information from video.

Advantages:

- **People count:** Applying face detection to a video captured at the incident site can help give a quick estimation of the number of people at the scene.

- **Minimal data:** The result of face detection can be as small as a single integer keeping data transmission to a minimum.

Disadvantages:

- **Different faces:** Faces captured on camera have varying features such as color, angle, shape and size. It is difficult to find all types of faces with a single face detection algorithm.
- **False positives:** The more types of face types you try to detect, the more objects can be mistaken for faces leading to a high the amount of false positives.
- **Limited information:** Having an estimate of the number of people at the scene is obviously just a small part of the information that is contained within the video. While good as a supplement to other information extracting techniques, face detection alone does not suffice.

Chapter 5

Related Work

In this chapter we look at work related to the focus of our research. Our main focus is the resource consumption of mobile devices and the measurement thereof. We divide the related work into two main categories: power consumption and bandwidth usage. Related work from within these categories are presented in Section 5.1 and Section 5.2 respectively.

5.1 Power Consumption

In research connected to resource limited networks such as MANETs, power consumption is always a factor that should be taken into consideration. There are two main ways to analyze power consumption: power measurement and power modeling.

5.1.1 Power Measurement

Power measurement is a quantitative method for analyzing power consumption. It involves physically measuring the amount of power used and provides statistical data that can be used for instance to approximate the lifetime of a battery.

In research, power measurement is widely used to analyze the power consumption of mobile phones while performing certain tasks, especially when comparing different protocols or approaches. A handful of papers focus on the difference in power consumption between various networking technologies Balasubramanian et al. [1] compares the power consumption when downloading using either WiFi or 3G. A similar comparison between 2G and 3G is conducted by Perrucci et al. [23].

Raghunathan et al. [25] measure the power consumption of separate subsystems used for communication, computation and storage. They compare the use of Bluetooth to the use of WiFi and measure the power consumption of several different actions such as transmission, processing and streaming of audio and video. Bluetooth generally has a lower power consumption than WiFi, but also a lower maximum data transfer rate. Their results show that streaming of moderate sized videos or high quality mp3 audio is not enough to saturate the bluetooth channel. Therefore they

achieve the same throughput as they do when using WiFi, but with a lower power consumption. However, with high quality video (high enough to saturate the bluetooth channel), WiFi would be the better choice.

Other work has focused on the power consumption of mobile applications, such as e-mail applications [20], video sharing applications [35] and cloud-based BitTorrent [14] services.

Most power measurement in the literature is done system-wide and not individually for each component. This is because one would need an overview of the circuits of the mobile device to do component specific power measurements. Such an overview is normally not available for commercial mobile phones. One notable exception is the FreeRunner mobile phone used by Carroll and Heiser in their work with component-level power modeling [4].

There are two main methods of obtaining system-wide power measurements. The first is through application programming interfaces (APIs) in the mobile operating system or energy profiling software. The other is through physical measurements with power meters. The ladder can also be used for component specific power measurement if one has the required information. Chandramouli et al. [5] and Zhang et al. [36] utilize such physical measurements.

Both of these techniques for physical power measurements have their advantages and disadvantages. While physical measurements with power meters can be done at a higher frequency, using software and operating system APIs is more feasible in a mobility scenario. However, not all mobile platforms have such APIs available. The choice of power measuring technique thus depends on the availability of instruments and the requirements for the experiment.

5.1.2 Power Modeling

Power modeling is a method for analyzing power consumption without performing physical measurements. There are two main types of methods for building a power model: deterministic methods and statistical methods.

Deterministic Methods

Deterministic power modeling involves mapping software operations to hardware activities and approximating the resulting power consumption. Most hardware components can operate in several power states, which correspond to different levels of power consumption. What power state a hardware component is in, depends on the activities currently being performed. Thus, based on the activities being performed, one can determine the current power consumption of a hardware component. Doing this for each hardware component provides a power model for the whole system.

Statistical Methods

Statistical methods can also be used to model the power consumption of hardware components. Statistical power modeling methods try to find the relation between power consumption and model variables based on statistical methods, such as linear regression.

5.2 Bandwidth Usage

Bandwidth usage is another important aspect of communication in a resource challenged network, such as a MANET.

In their paper, Kristiansen et al. [15] conducts real-world experiments to study the forwarding capabilities of handheld devices. In research related to streaming video over MANETs, it is common to utilize network simulators such as ns-3[1] as opposed to conducting real-world experiments. However, most network simulators do not take node resources into account. Since the limited resources of the mobile devices can become a bottleneck, it is important to conduct realistic real-world experiments. We can draw an analogy between this and the measurement of power consumption. Though there are many models for power consumption, it is close to impossible to achieve the same accuracy as real-world measurements. Kristiansen et al. analyze the resource consumption of a Nokia N900 while streaming video. Through their experiments they are not able to achieve a throughput close to the one supported by the wireless medium. The reliability of results of simulations based only on the capability of the wireless medium, without taking node resources into account can therefore be questioned.

The paper from Halvorsen et al. [13] is also related to the bandwidth usage of mobile devices. Here they set up a two-hop MANET consisting of three Nokia N770 devices. Their results show that the intermediate node is capable of forwarding 12 streams with a bitrate of 200 Kbps, six streams with a bitrate of 500 Kbps or three streams with a bitrate of 1000 Kbps, with an acceptable video quality.

[1] www.nsnam.org

Chapter 6

Design

In this chapter we go through the design of our solution. We begin by stating its objectives in Section 6.1. We then describe the design of an application intended for the CCC personnel in Section 6.2. This application incorporates the information conveying techniques presented in Chapter 4 within separate components. Finally we go through the system design for the part of the system we implement for use in our experiment in Section 6.3.

6.1 Objectives

In this section we present the objectives of our solution for conveying information based on video clips from the incident site to the CCC.

6.1.1 Information Availability

Ideally, the workers at the CCC should be in possession of all relevant information allowing them to make important decisions away from the hectic environment of the incident site. A substantial amount of this information is based on video from the head-mounted cameras of rescue workers on site. We aim to make all relevant information available. Our solution must thus be able to avoid loss of important information. The objective of information availability leads to other objectives such as low power consumption to ensure that the devices in the network do not run out of power too fast and avoiding network congestion to avoid information being lost due to packet loss.

6.1.2 Low Power Consumption

In a network of mobile devices, limited power is an aspect that must be considered. The source devices, carried by rescue workers with head-mounted cameras are especially important. Their batteries being depleted would be the end of all conveying of information from the rescue worker's camera at the incident site to the CCC. A low power consumption for

the mobile devices in the network is therefore one of the objectives when designing our solutions.

6.1.3 Minimizing Packet Loss

We want a minimum of packet loss in our network. Packet loss can be caused by network congestion, bad quality links due to mobility or packet collisions among other things. Network congestion takes place when a node carries so much data that its quality of service is reduced. We want to minimize data transfer in the network to reduce packet loss due to network congestion and collisions. Packet loss due to bad quality links is avoided by the DTS-Overlay which stores packets that cannot be forwarded in a buffer for later transmission.

6.1.4 Understandable Information

We want to present information in such a manner that it can be understood quickly and easily by the personnel at the CCC. Looking through long videos would be both tiring and time consuming and by the time the personnel discover important information, that information might already be outdated. Utilizing panoramic images is a good way to give an overview of an area and to sum up a video segment, either as a replacement for or as a supplement to the video segment itself.

6.1.5 Timely Delivery

Even if the network is delay tolerant, the personnel at the CCC should get information as fast as possible. An objective of our solution is thus to deliver the information in a timely manner. By lowering the amount of data to transfer we aim for a decrease in total time from when the video is received at the source device to when the information from the video is available at the destination.

6.2 CCC Application

In this section we propose an application that the personnel at the CCC could use to get an overview of the situation at the incicdent site, utilizing the aforementioned information conveying techniques. A sketch of what such an application might contain can be seen in Figure 6.1. The sketch shows a group of tabs, each containing information related to a single camera. For the selected camera you can see the following elements:

- Episode Timeline
- Key Frame Viewer
- Episode Player
- Panorama Viewer

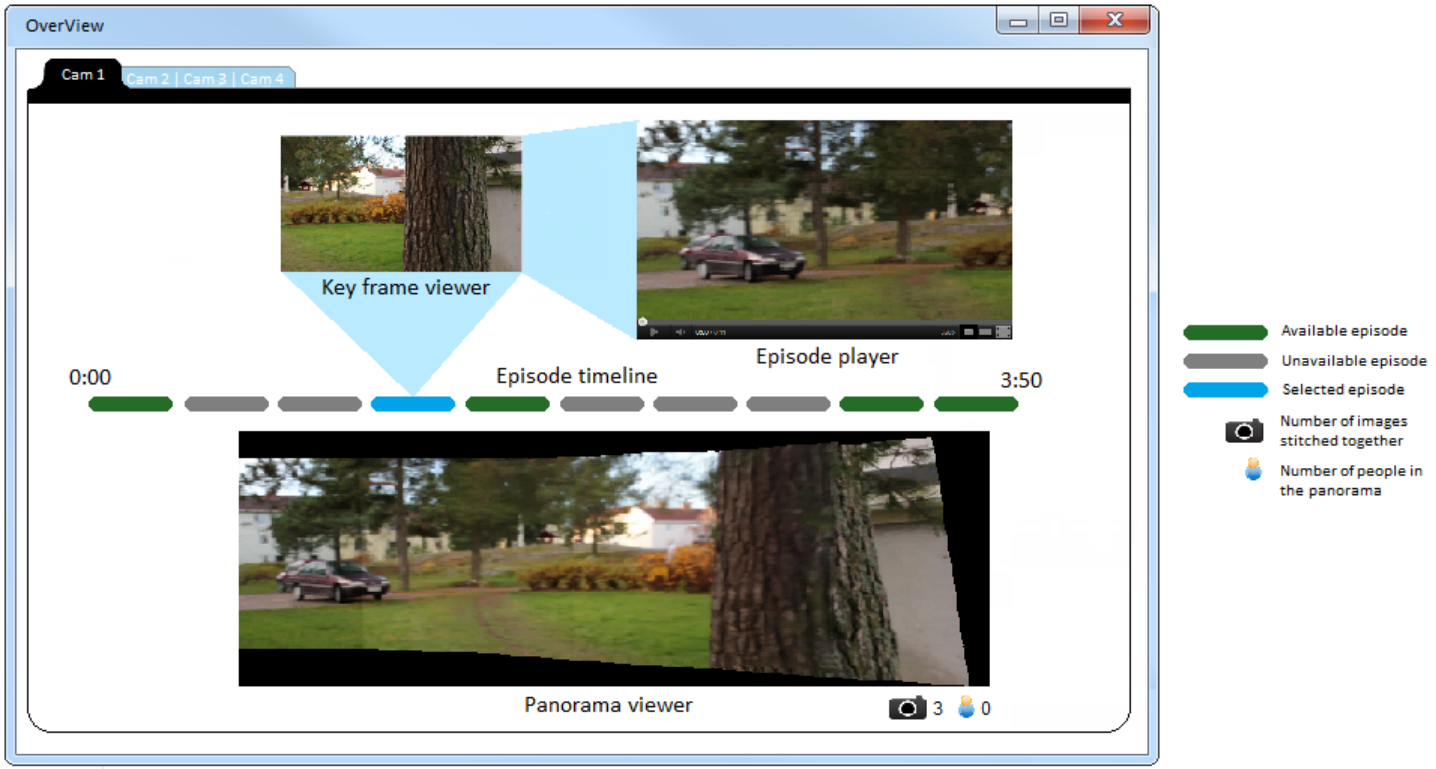


Figure 6.1: A sketch of the application for the personnel at the CCC

Episode Timeline

In the middle of the application sketch, the episodes from which the CCC has received key frames are listed in chronological order. Green bars represent episodes that have been fully received by the CCC and are ready to be watched. Gray bars represent episodes from which the key frame has been received, but not the actual footage. The blue bar is the currently selected episode. The purpose of the episode timeline is to give an overview of what footage is already available, what footage can be requested and the order of happenings at the incident site.

Key Frame Viewer

When selecting an episode from the episode timeline, the key frame viewer displays the key frame representing that episode. The key frame is meant as a summary of the episode and should give an idea of what you can see in that specific episode. When selecting a gray episode, an option to request the whole episode becomes available next to the key frame viewer.

Episode Player

When selecting a green episode from the episode timeline, an episode player appears in addition to the key frame viewer. Using this player, the personnel at the CCC can watch the whole episode.

Panorama Viewer

At the bottom of the sketch you can see a panorama of the area created by concatenating frames from the video using image stitching, as described in Section 4.4. The purpose of the panorama is to give a better overview of the area. Beneath the panorama, the number of images it consists of and the number of people detected within it are listed. The number of people is obtained by using face detection, as described in Section 4.5.

6.3 System Design

The focus of our work is the image stitching related component of the proposed CCC application. We have chosen this part of the application because a panoramic image, obtained by performing image stitching, provides important information without requiring a lot of bandwidth. Image stitching is, however, a processor heavy task. When evaluating the image stitching approach we therefore investigate if power saved on transmission makes up for the extra power consumption from the image stitching operation.

In this section, we describe the design of our system. We go through the system requirements for our image stitching approaches and the video transmission approaches we compare them to. We then describe the header for packets sent through the network.

6.3.1 Requirements

We conduct an experiment to investigate the effects of different approaches for utilizing image stitching when conveying information from a video. We compare the results of our image stitching approaches to more standard information conveying techniques such as transmission of uncompressed or compressed video. As a prerequisite for this experiment, our mobile device needs to be capable of performing a set of tasks. As our mobile device, we use the Android device Samsung Galaxy Nexus as seen in Figure 6.2. We choose this device because it is a state of the art smart phone and it runs the newest version of the Android operating system: Ice Cream Sandwich (ICS). We begin by defining the set of tasks that our Android device must be capable of performing:

- Transmission of files to a remote host
- Compression of video
- Extraction of images from video
- Stitching of images
- Forwarding

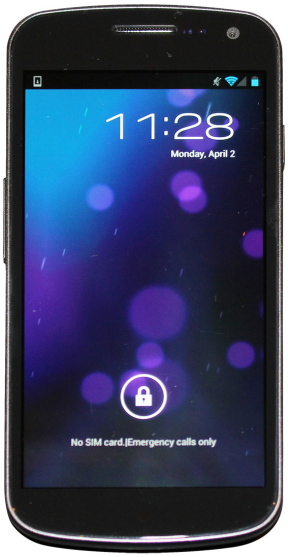


Figure 6.2: The Samsung Galaxy Nexus

File Transmission

With the main requirement for the application scenario being transmission of information from the incident site to the CCC, we need a way to transfer files. The files in question are either uncompressed video, compressed video, frames extracted for stitching or a stitched image depending on the

configuration. In our experiment the files will be sent from the Android device to a computer.

For file transmission, the two main transport layer protocols are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). TCP is a connection oriented, reliable transmission protocol which ensures that packets arrive and that they arrive in the right order. In our application scenario, using TCP is not feasible as our target network is disruptive, meaning that it is not likely to have a complete path from the source to the destination.

UDP is a connectionless, unreliable protocol, meaning packets are sent out on the network without knowing if the receiver is actually there listening for them. When using UDP, there are no acknowledgements (ACKs) sent to the sender when receiving a packet. This means there is a chance of packet loss and packets may arrive in the wrong order. UDP assumes that error checking is not necessary or is done in the application. As we want to receive most packets in a timely manner rather than all packets eventually, UDP is preferred for our application scenario.

The deficiencies of UDP are handled in the DTS-Overlay implemented by Lindeberg et al. [16]. Normally, when the transmission of a packet fails, there are several retransmission attempts performed at the MAC layer. If the packet has not been successfully sent when reaching the retransmission limit, which is normally seven, the packet is dropped. Instead of dropping the packet when the maximum number of retransmissions is reached, the solution of Lindeberg et al. is to make the MAC layer return the packet to the DTS-Overlay for later transmission. Lindeberg et al. has also shown how lowering the maximum number of retransmission attempts at the MAC layer can help avoid unnecessary transmission over low quality links. For instance, if the quality of a link has degraded due to the devices moving out of range of eachother, there is no point in trying to retransmit packets over that link multiple times. As future work it would be natural to implement our approach in conjunction with the DTS-Overlay and it is thus interesting to investigate transmission using UDP in this experiment.

In their paper, Kristiansen et al. [15] studies the capabilities of modern handheld devices to forward video streams. Though they work with the Nokia N900 their results should be applicable to our setup using the Samsung Galaxy Nexus. They find that the devices can handle five concurrent streams with a bit rate of 1 024 Kbit/s. We therefore use a transmission speed of 1 024 Kbit/s in our experiment. To achieve the desired bitrate when using UDP we make the client wait a short period of time between transmission of packets. The packets we send are 1 500 B which consists of 1 472 B data and 28 B headers. 1 024 Kbit/s equals 128 000 B/s. With a payload of 1 472 B we thus have to send $\frac{128000}{1472} = 86.96$ packets per second. To achieve a maximum transmission speed of 1 024 Kbit/s we therefore have to wait $\frac{1}{86.96} = 0.0115$ seconds or 11.5 milliseconds between transmission of packets. To do this, we utilize the Java *sleep* function, but since this does not accept decimal numbers, we make our client wait 11 milliseconds between each packet. This means that we achieve a maximum transmission speed slightly higher than 1 024 Kbit/s. The maximum

transmission speed is $8 * 1472 * \frac{1}{0.011} = 1071$ Kbit/s to be exact.

Compression

Our Android device must be able to convert the .mov file received from the camera of the rescue worker to a .mpg file using MPEG compression as described in Chapter 4. Video compression is a CPU intensive task and thus consumes power, but at the same time it reduces the size of the video so that subsequent transmission will consume less power than transmission of uncompressed video would.

Image Extraction

To be able to create a stitched image based on a video clip, our Android device must first extract images from the video. We extract frames at a fixed rate which is based on the speed of the swiping motion in the video. Since successful image stitching relies on a certain degree of overlap between the images that are to be stitched, frames must be extracted more often during a fast swipe than during a slow swipe.

Image Stitching

Our Android device needs to be able to perform image stitching as explained in Chapter 4. For this purpose we require a program running on the Android device that can create stitched images using images extracted from video as input.

Forwarding

The last main task that the Android device must be capable of performing is to forward files received by other nodes in the network. There are four main types of nodes in our network. We have the *source devices* belonging to rescue workers with cameras, the *message ferry devices* belonging to emergency personnel traveling back and forth between the incident site and the CCC and the actual CCC node. The fourth and final type of node to which most devices belong, is the *forwarding device*. The job of the forwarding devices is to forward packets received from a source device or another forwarding device along the path to the CCC. In most cases this implies forwarding it to or on the path to a message ferry, which stores and carries the packet until it comes into range of the network partition containing the CCC.

6.3.2 Packet Header

Packets transmitted through the network originate from various cameras and depending on the configuration, the packets may be of different types. To be able to differentiate between them when they arrive at the CCC, we utilize a packet header. The first field of this header is the *camera id*, showing which camera captured the video that this packet is related to.

The next field is the *packet type*, represented by a character. The possible packet types are T (Transmission) for packets from uncompressed videos, C (Compression) for packets from compressed videos, E (Extraction) for packets from extracted images and S (Stitching) for packets from stitched images. We also include a *time stamp* field in the header. For type T and type C packets, the time stamp identifies the time in the video of the frame which the packet is part of. The camera id combined with the time stamp allows us to correctly order packets, should they arrive out of order. If multiple configurations are used at the same time, different priorities can be given to different packet types allowing the most important packets to arrive first. Such a priority scheme can help negate the effects of network congestion for the most important packets.

6.3.3 Lightweight Applications

To adhere to the objective of low power consumption, we make the applications that perform the required tasks as simplistic as possible. This avoids unnecessary consumption of power connected to having an advanced user interface or visual effects. The goal of these applications is not to look good, but to perform required tasks while consuming as little power as possible.

Chapter 7

Implementation

In this chapter we discuss the implementation of programs and applications for use in our experiment. In Section 7.1, we present the system architecture. We take a look at each individual component, what functionality they provide and how they do it as well as how data flows in the system. Section 7.2 lists the technologies, libraries and programs used to implement the various components.

7.1 System Architecture

In this section we go through the architecture of the system we use in our experiment, describe its components and look at the system flow. Figure 7.1 shows the architecture of our system and gives an overview of the different components. We have omitted the message ferry device in this figure, but it contains the same components as a forwarding device with the addition of a storage buffer for packets that can not be forwarded immediately.

7.1.1 Components

In this section we go through the components needed for our image stitching approach, as well as additional components needed to perform our experiment.

Transmitter

As described in Chapter 6 we need a program or an application for transmitting files. This Transmitter must be able to take a single file or multiple files as input, split them into packets, add the packet header and transmit the packets to a remote host. The Transmitter component is used by the source device.

The Transmitter we implement gets the filename or -names and the number of files to transmit from the command line and transmits the file(s) to the remote host. Algorithm 7.1 shows the pseudo code of the Transmitter component. In our experiment we do not add the packet header mentioned

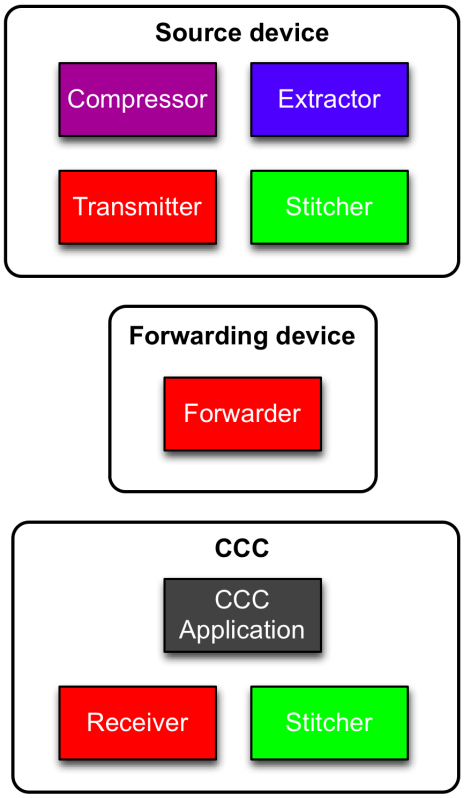


Figure 7.1: The system architecture

in Chapter 6. A packet size of 1 472 bytes makes the packets 1 500 bytes when the IP header is added.

Receiver

At the receiving end we need a program or an application that can receive the files sent by the Transmitter and save them locally. This Receiver needs to be able to receive packets, process and strip the packet header, assemble the packets to create a file and finally save the file. The Receiver component is used at the computer at the CCC. Algorithm 7.2 shows the pseudo code of this component.

Compressor

We need a program or an application capable of compressing video. In particular this Compressor must be able to take a QuickTime File Format (.mov) video and compress it to an MPEG video (.mpg). The Compressor component is used at the source device.

Extractor

The Extractor is a program or an application that extracts frames from video and saves them as JPEG images. We need to be able to select the rate at

Algorithm 7.1 Transmitter pseudo code

```
for all files do
    bytesLeft = filesize
    while bytesLeft > 0 do
        if bytesLeft > 1472 then
            Copy 1472 bytes from file to a packet
            Send the packet to specified IP and port
        else
            Copy the rest of the bytes to a packet
            Send the packet to specified IP and port
            bytesLeft = 0
        end if
    end while
end for
```

Algorithm 7.2 Receiver pseudo code

```
outputFile = null
while true do
    Listen to traffic on predefined port
    if outputFile = null then
        Create new outputFile
    end if
    Receive a packet
    Write its content to outputFile
    if packetSize < 1472 then
        Close outputFile
        outputFile = null
    end if
end while
```

which images should be extracted from the video.

Stitcher

We require a program or an application that can stitch images. Taking an arbitrary number of overlapping images as input, this Stitcher should output a panorama image which is a combination of the input images. The Stitcher must adhere to the image stitching algorithm presented in Section 4.4.6 or the equivalent of it.

Forwarder

The forwarder receives packets from one device and forwards them to another device. To this effect it is a combination of the Receiver and the Transmitter. It must be able to receive a packet, check the destination IP address in the IP header, select where to forward the packet based on information from its routing table and finally transmit the packet to the next hop. For our experiment we create a simple version of the Forwarder that forwards all packets received to a predefined IP address. The reason for the simplification is that the DTS-Overlay is normally responsible for the routing and that our main concern is the power consumption when forwarding a file. The Forwarder component is used by all forwarding devices. Algorithm 7.3 shows the pseudo code of the Forwarder component.

Algorithm 7.3 Forwarder pseudo code

```
while true do
    Listen for traffic on predefined port
    Receive packet
    Send packet to next hop
end while
```

CCC Application

The CCC Application, described in Chapter 6, is an application intended for the personnel at the CCC. We do not implement the actual CCC application in this work. The main focus of this thesis is the measurement of power consumption and bandwidth usage of getting a stitched image ready for the CCC application. In future work we would like to implement the actual CCC application as well.

7.1.2 Inter-Component Communication

In our implementation aimed for use in the experiment, there is barely any communication between the components. The components are called in a set order by shell scripts depending on the desired result. For instance if we want to transmit a compressed video the Compressor component is

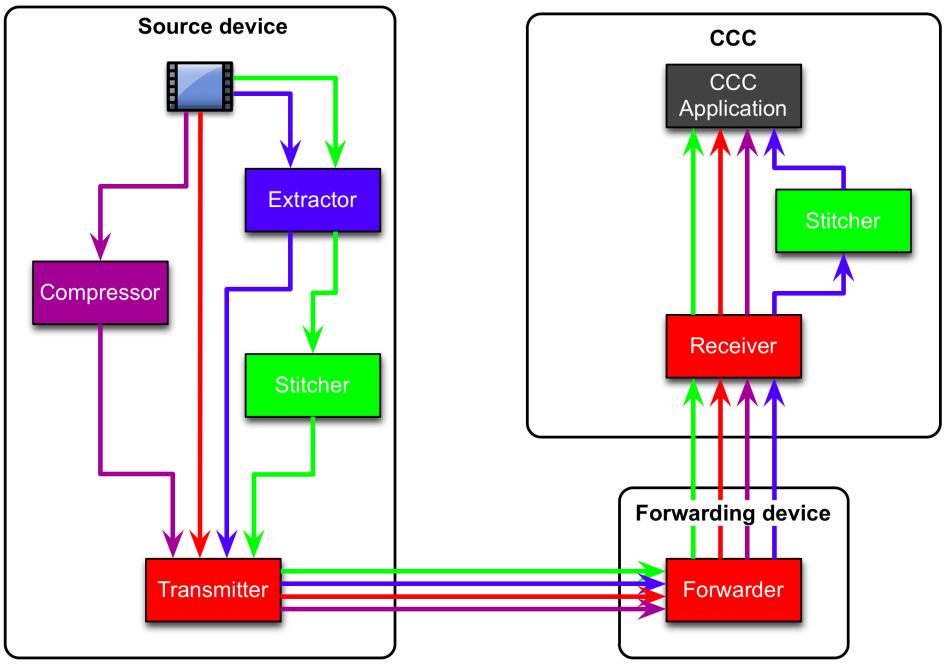


Figure 7.2: Data flow in the system

called first and the Transmission component thereafter. As future work we would like to implement more communication between the components and make this process more transparent to the user.

7.1.3 Data Flow in the System

Figure 7.2 shows the flow of data in our system. There is only one forwarding device in this figure, but in practice there could be *n* forwarding devices between the source and destination as well as a message ferry device. The difference between a forwarding device and the message ferry device is that the ladder contains a buffer for storing packets that can not be forwarded until later. The reason they cannot be forwarded immediately could be that the message ferry device is not within range of the next hop. When the message ferry moves within range of the next hop, the packet will be forwarded. On the left hand side of the figure, we see the source device with its components. In our experiment we explore four routes through the system, from the uncompressed video at the source device, through one or multiple forwarding devices to the result at the CCC.

The red line shows the path through the system taken when transmitting uncompressed video. The uncompressed video is transmitted by the Transmitter, forwarded by the Forwarder at each of the n forwarding devices and received by the Receiver within the computer at the CCC. In this case the CCC ends up with the whole uncompressed video.

The purple line shows the uncompressed video being compressed by the Compressor before being transmitted by the Transmitter, forwarded by the Forwarder(s) and received by the Receiver. In this case, the CCC obtains

a compressed version of the original video.

For both the green and blue lines, the end product at the CCC is a panorama image. Both these routes starts with frames being extracted by the Extractor at the source device. They only differ in where the stitching is performed. When taking the blue route, the extracted images are transmitted by the Transmitter, forwarded by the Forwarder(s) and received by the Receiver before being stitched by the Stitcher at the CCC. For the green route, the stitching is performed by the Stitcher at the source device before transmitting, forwarding and receiving the stitched image.

7.2 Technologies Used

In this section we present the technologies, libraries and programs used within our system.

7.2.1 Java Sockets

For transmitting files between the Transmitter, Receiver and Forwarder components we utilize socket communication. We implement a client and server based on Liu Feipeng's implementations[1] and altered to suit our needs. We implement the client both as an Android application and as a normal Java desktop program and the server only as a Java desktop program. In addition we implement a forwarding application for Android that is a combination of a client and a server. The forwarder listens for traffic on a specific port as a server. When receiving packets the forwarder starts a client which transmits these packets to another node. There are several ways to implement forwarding. We could receive the whole file before starting to forward it or forward every single packet individually. We could perform the forwarding at the IP layer or at the application layer of the OSI model. In our experiment we have chosen to forward packets individually and perform the forwarding at the application level. We have chosen this because it is similar to how it is done when using the DTS-Overlay.

The Android client application and the forwarding application can be started like any other applications by clicking their icons on the home screen. In our experiment, however, we start them by calling the Android Activity Manager (am) from the command line. The exact commands for starting a client and a forwarding instance from the command line can be found in Appendix B.

In socket communication, the server instance creates a server socket and listens for incoming connections on a specific port. A client connects using the IP address of the device acting as a server and the port number which it is listening on. The server then accepts and the connection is established. The foundation for communication is now in place. The next step in our implementation is transmission of files.

[1] http://www.roman10.net/android-udp-client-and-server-communication-programmingillustrated-with-example/

For the part of the experiment where the Android device acts as the source device, the server runs on the laptop, listening for incoming connections and packets on a specific port, while the client runs on the Android device. When forwarding, both a server and a client run on the laptop. The Forwarder application runs on the Android device and "forwards" packets from the laptop back to the laptop.

7.2.2 Android Implementation and Integration

Since some of our components are based on programs and libraries created mainly for being used on a computer, a large part of our implementation consists of porting and integrating these components so that they will work on an Android device. For the components that transmit and receive data, the Transmitter, Forwarder and Receiver, we implement socket communication for the Android device, as mentioned in the previous section.

The differences in how Java code is written for Android and for a computer are minimal. Instead of a *main* method as we have in regular Java, we have an *onCreate* method that is executed when the application is run. The class containing this method needs to extend the *Activity* class. Command line arguments are handled differently as an application is started from the command line using Android Activity Manager (am) and not by using the *java* command as you would on a computer. The command line arguments provided are key value pairs and are stored in a bundle. Algorithm 7.4 shows how to access these command line arguments.

Algorithm 7.4 Obtaining command line arguments

```
Bundle extras = this.getIntent().getExtras();
if (extras != null) then
    if (extras.containsKey("file")) then
        filename = extras.getString("file");
    end if
end if
```

When coding for Android we have to set permissions for our applications. These permissions are set in the *AndroidManifest.xml* file of the application. For instance the manifest file needs the following permission for the application to be able to access the network which is a requirement to send or receive packets:

```
<|uses-permission android:name="android.permission.INTERNET"/|>
```

Android also requires that networking operations is not run on the main thread of the application. This is to ensure responsiveness for the user.[2] Except for this handful of idiosyncrasies, programming in Java for Android is much like programming in Java for a computer. In most cases you can use an existing Java class as long as you create an activity with an *onCreate*

[2] http://developer.android.com/guide/practices/design/responsiveness.html

method that creates an instance of and calls methods of the existing class. Obviously Java for Android has a lot of other principles and possibilities, but to make simple applications like the ones we make for use in our experiment, this is most of what we need to be aware of. The Android developer site is an invaluable asset[3] when implementing applications for Android.

Compression and extraction is done by FFmpeg. This is done by executing commands directly from the command line or from a shell script. Since the stitching module of OpenCV is written in C++, the Stitcher application calls it through a native function using Java Native Interface (JNI). When stitching using the computer at the CCC we can call the OpenCV stitching module directly from the command line. When the stitching operation is completed, the Stitcher applications starts a Transmitter instance to transmit the resulting image.

7.2.3 FFmpeg

Our Compressor requires a way to compress video and our Extractor a way to extract frames from video. FFmpeg provides both of these functionalities. It is an open source multimedia framework used for recording, converting and streaming of audio and video. With regards to compression, FFmpeg does all of the work for us internally. To compress a .mov file to a .mpg file we just have to provide the name and format of the input file and the output file. Extraction works much in the same way. To extract frames from a video , we simply tell FFmpeg the name of the video we want to extract frames from, the number of frames per second we want to extract and what to save the extracted frames as. We have built FFmpeg for Android following the steps found at the blog "$\lim_{n\to 0} \frac{1}{n} = \infty$" [4]. The specific commands for compression and image extraction using FFmpeg can be found in Appendix B.

7.2.4 OpenCV

To perform the stitching of frames extracted from video we use OpenCV, an open source library of programming functions for real time computer vision[5]. For stitching at the source device, we implement an application based on the Pano application[6] which we strip down and alter to only perform the actual stitching operation. This is done to avoid spending resources on non-vital elements such as a user interface. Our application takes the name of the video and the number of images to stitch as input parameters and outputs a panoramic image obtained by stitching the images. To perform the actual stitching operation our application, like the Pano application, utilizes the stitching module of OpenCV. This module adheres to papers by Shum and Szeliski [27], Uyttendaele [29] and Brown

[3] http://developer.android.com
[4] http://demo860.blogspot.com/2010/06/ffmpeg-libx264-build-for-android.html
[5] opencv.willowgarage.com
[6] http://code.google.com/p/android-opencv-panorama/

and Lowe [3]. To be able to stitch images on our Android device, we have ported OpenCV to the Android operating system, following the guide at opencv.itseez.com[7].

For stitching on the computer at the CCC we can utilize the OpenCV stitching module directly, by calling it from the command line. We go through the specific stitching commands and the meaning of the various parameters in Appendix B.

[7] http://opencv.itseez.com/doc/tutorials/introduction/android_binary_package/android_binary_package.html

Chapter 8

Evaluation

In our problem statement we stated that we want to give the CCC the best possible overview of the emergency situation, while preserving the scarce resources of the network. The solution we pursue is to give this overview in the form of stitched images based on video input from the camera of a rescue worker at the incident site. In this chapter, we evaluate the effects of image stitching as an alternative to video streaming with regards to resource usage and the lifetime of the devices in the network. To do this we conduct an experiment in which we compare different image stitching configurations to transmission of uncompressed video and compressed video. In the following sections we state our goal, describe our setup, present the configurations, define the workload, select our metrics and discuss the results before summarizing the experiment.

8.1 Goal

Our goal for this experiment is to find the optimal solution with regards to what should be transmitted between the incident site and the CCC when using an image stitching approach for conveying visual information. We also aim to determine the best location for performing the image stitching operation. To accomplish these goals we compare image stitching configurations to the transmission of uncompressed or compressed video. We investigate whether or not the potential decrease of data transmitted from the mobile devices in the network makes up for the increase in processing time and related resource usage. Additionally, we investigate what impact changing the location of the image stitching task has, i.e. whether the stitching operation is done at the source device or at the CCC.

8.2 Setup

The physical setup for our experiment, as depicted in Figure 8.1, consists of a mobile device, a laptop, a power supply and a stationary computer. In this section we begin by describing these components and how they are connected and interact. We go on to describe the more virtual setup

including software and device settings.

Figure 8.1: The experiment setup

8.2.1 The Mobile Device

As mentioned in Chapter 6, we have chosen the Samsung Galaxy Nexus, a smart phone running the Android Ice Cream Sandwich operating system, as our mobile device. During an experiment, the mobile device is used to run the tasks required by the different configurations. In the first half of the experiment, the mobile device takes on the role as the source device, while in the second half it acts as a a forwarding device.

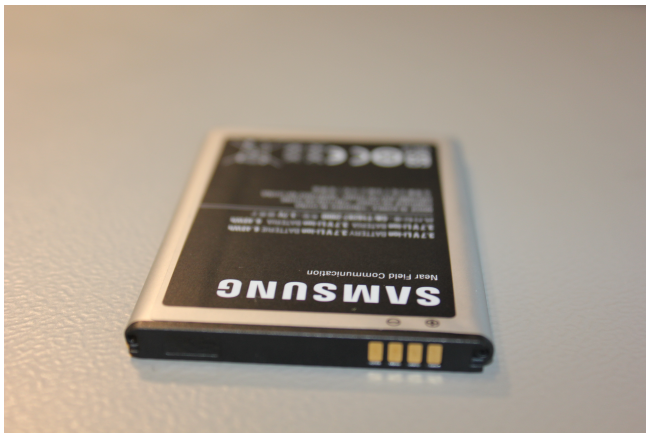


Figure 8.2: The battery of the Samsung Galaxy Nexus

The Samsung Galaxy Nexus has a dual-core 1.2 GHz processor. The battery is a 3.7 V lithium ion battery with an effect of 1750 mAh. This

means, for instance, that a fully charged battery can power the device for 17.5 hours if the device on average is consuming 100 mA. The battery, as seen in Figure 8.2, has four terminals of which we utilize the first three. The positive and negative terminals are labeled with a + and a - respectively. The terminal in between these is the *thermistor*. A thermistor or thermal resistor is used for temperature sensing and compensation. It has a varying resistance based on the temperature. If the resistance measured at the thermistor is not within the correct range, the phone will not boot. The battery normally supplies the required resistance. When removing the battery, we therefore need to introduce a resistance within the correct range and connect it to the thermistor terminal to allow the phone to boot. We do this by connecting a resistor with a resistance of 80 kΩ between the thermistor terminal and the negative terminal of the device.

8.2.2 The Laptop

The laptop is the peer of the mobile device in our experiment. When the mobile device takes on the role as the source device, the laptop takes on the role as the CCC, in that it receives the files sent by the mobile device. Figure 8.3 illustrates this. When the mobile device acts as a forwarding device, the laptop is both the sender and receiver of data, as seen in Figure 8.4. The laptop has a quad-core 2.2 GHz processor.

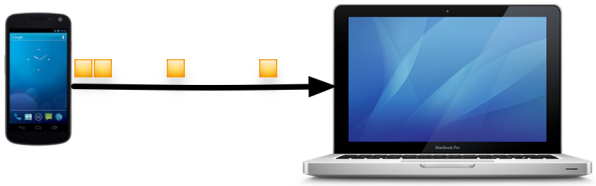


Figure 8.3: Using the Android device as the source node

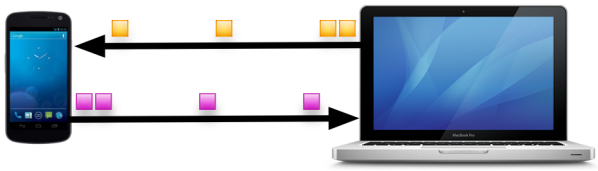


Figure 8.4: Using the Android device as a forwarding node

8.2.3 The Power Supply

As power supply we use the Agilent (Hewlett-Packard) E3631A Triple Output DC Power Supply as seen in Figure 8.5. In addition to supplying the Android device with power, the power supply measures the amount of power the device uses. It also enables us to set the voltage and a limit for the maximum amount of amperes the device is allowed to use. We set the voltage to 5 V, which is the same voltage as when the device is charging with a USB cable. The ampere limit is set to 2.3 A, which is more than enough for our use.

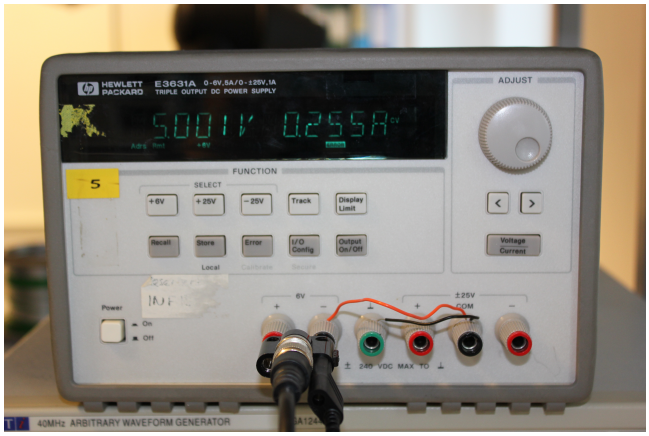


Figure 8.5: The power supply

8.2.4 The Stationary Computer

The power supply is connected to a stationary computer to allow us to extract measurement results from it. The stationary computer is only used for this purpose and does not take part in the network with the laptop and the mobile device.

8.2.5 The Network

To avoid collisions and packet loss due to other traffic, we set up an ad-hoc network between the mobile device and the laptop. Instructions on how to do this can be found in Appendix A.

8.2.6 Connecting the Mobile Device to the Power Supply

To connect the Android device to the power supply we first remove the battery. To power the device, we connect the negative terminal and positive terminal to the negative and positive terminals of the power supply. Figure 8.6 shows how the wires are connected to the battery terminals within the device. The gray object in the right part of the image is the resistor with a resistance of 80 kΩ. The red and black wires connect the positive and negative terminals of the device to the corresponding terminals of the

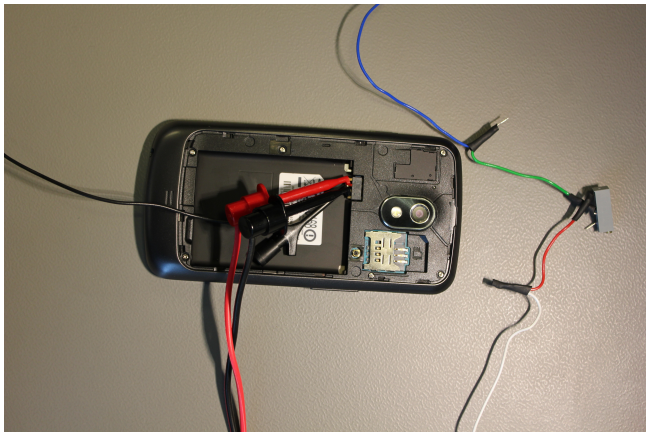


Figure 8.6: The wiring

power supply. Figure 8.7 is a circuit diagram showing how the Android device is connected to the power supply.

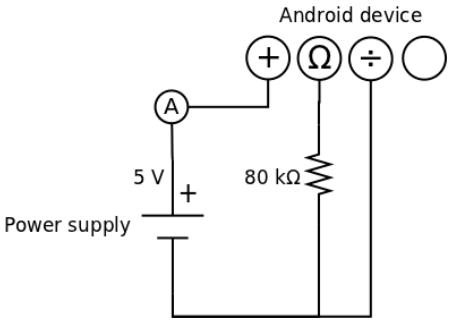


Figure 8.7: A diagram of our setup

8.2.7 Virtual Setup

In addition to the physical setup of devices and hardware, we have the more virtual aspect of setting up our experiment. Some of the following steps are needed for the experiment to work, some are for logging purposes and some are measures taken to prevent noise and variations in the experiment conditions between each run. The virtual steps of setting up the experiment is:

- Setting up the network
- Fixing the CPU frequency
- Start logging power consumption
- Log network traffic
- Start CPU usage monitoring
- Start UDP server(s)

We first set up an ad-hoc network using the Android device as a portable WiFi hotspot and connect to its network from the laptop. We then fix the CPU-frequency of the device to 1.2 Ghz to utilize the full potential of the CPU and to make sure all configurations compete on equal terms. Details on how to set up the ad-hoc network and fixing the CPU frequency for the Samsung Galaxy Nexus can be found in Appendix A.

Once this is in place, we start various logging. To obtain information about the power consumption we run a MATLAB[1] script on the stationary computer, querying the power supply at a fixed interval. In our experiment we query the power supply every 0.1 seconds. We log CPU usage by running the Linux command *top* on the Android device and network traffic by running *tcpdump*[2] on the laptop saving the output of each command to separate files. It is important to note that running top on the Android device is an additional source of power consumption. It should, however, affect each configuration equally.

The next step is starting server instances for receiving files. When using the Android device as the source device, we start a server instance on the laptop. When using it as a forwarding device, we start one additional server instance on the Android device for files that are to be forwarded. Only when these steps are completed, we can start a run of the experiment. The subsequent steps are configuration specific.

The color displayed by the screen greatly impacts the power usage. From a power consumption of around 150 mA when displaying an entirely black screen to a power consumption of around 380 mA for a completely white screen. To lower the power usage and make the runs as equal as possible, all runs are started from the terminal, which displays white text on a black background. This is not completely fair as some runs will have more white output text than others, but it is as close to complete fairness as we can get. Some noise and variations are avoided in a more manual fashion. For instance, every few minutes the device pops up a message about a new update to the operating system. This causes the power consumption to spike for a small period. We avoid this variation simply by redoing the runs where this occurs. Each run is started by running a shell script from the terminal that in turn executes other scripts, commands or applications to perform the required operations.

8.3 Configurations

We investigate how different approaches to getting information from a video captured at the incident site to the CCC affect the resource usage

[1] www.mathworks.com/products/matlab/
[2] http://www.tcpdump.org/

of the mobile devices. The different configurations for our experiment are:

1. **Transmission (T)**: Transmit the video to the CCC in the format received from the camera.
2. **Compression (C)**: Compress the video captured by the camera and transmit the compressed video to the CCC.
3. **Extraction (E)**: Extract images suitable for stitching from the video and transmit them to the CCC. Then perform the stitching at the CCC.
4. **Stitching (S)**: Extract images and perform the stitching at the source device. Then transmit the stitched image to the CCC.

8.3.1 Transmission

The *Transmission* configuration can be seen as the baseline we compare the other configurations with. When using this configuration, we transmit video as it is when received from the camera, without any additional compression or other preprocessing. The amount of data transferred across the network when using this configuration is thus the size of the file we get from the camera multiplied by the number of hops from the source to the destination. In our experiment, the uncompressed video files received from the camera is in the QuickTime File Format (.mov). Such a file is a multimedia container file, containing one or more tracks, each being one of the following data types: audio, video, effects or text. Our files each contain an h264 encoded video track and an aac encoded audio track.

8.3.2 Compression

When using the *Compression* configuration, we compress the QuickTime File Format video using MPEG compression, as described in chapter 4. The reason we perform compression is to decrease the file size, thus decreasing the network load. We choose MPEG compression because it is a widely used compression technique that achieves a high degree of compression while still providing sufficient video quality. After compression we transfer the compressed video in the MPEG format from the Android device to the laptop. Normally, compression and transmission are done in parallel, but we choose to separate the two operations. This to be able to examine their power consumption separately and to better compare the compression configuration with the other configurations which performs transmission and other tasks separately as well. This may have an impact on the results in that runs take longer than if transmission and other operations were done in parallel.

The total payload when using the Compression configuration is the size of the compressed video multiplied by the number of hops. The source device consumes resources while compressing and transmitting the video, while the main source of resource consumption for the forwarding nodes is the forwarding of the compressed video.

8.3.3 Extraction

The idea of the *Extraction* configuration is to both lower the amount of data transmitted over the network and to decrease the processing required at resource challenged devices. Lowering the amount of data sent over the network is done by extracting selected frames from the video and transmitting them instead of the video itself. Decreasing the processing time at the mobile devices is achieved by performing the processor heavy task of image stitching at the location where resources and processing power is not an issue, namely the CCC. In this configuration, the responsibilities of the source device is to extract frames suitable for stitching from the video and transmit these images along the path to the CCC. Other nodes in the network once again work as forwarding nodes and their resource usage is mainly related to forwarding the selected frames. The computer at the CCC performs the actual stitching task. At the CCC, power usage is not an issue and the additional processing power of the computer compared to the Android device should make the stitching operation even faster. The amount of data transferred in this case is the combined size of the extracted images multiplied by the number of hops between the source device and the CCC. The source device consumes resources in connection with extraction and transmission of images, while forwarding nodes consume resources when forwarding these images.

In our experiment, we are using a fixed rate when extracting frames from the video, depending on the speed of the swipe. From three of the videos we extract one frame per second, whereas from the last video, we extract one frame every two seconds. This is because of the slower swiping motion in the last video. Using a fixed rate is not the optimal way of selecting frames for stitching. An intelligent algorithm for selecting the next frame based on the amount of overlap with the previously selected frame would be preferable. If this study shows us that image stitching reduces the amount of data being transferred across the network even with a suboptimal approach for frame selection, then we can be certain that image stitching is feasible and leave developing better algorithms for frame selection as future work.

8.3.4 Stitching

The goal of the *Stitching* configuration is to minimize the amount of data that has to be transmitted. In a delay tolerant network with nodes coming and going and variable uptime of links, transmitting as little data as possible can prove beneficial. In this case, the source device extracts images from the video and performs the stitching operation. The stitched image is then transmitted along the path to the CCC. The amount of transmitted data is the size of the stitched image multiplied by the number of hops. The resource usage at the source device is related to extraction of images, stitching and transmission of the resulting image, while other nodes in the network consume resources when forwarding the stitched image. On one hand the resource consumption related to processing is high at the source

device. On the other hand, as the stitched image is smaller than the video segment it presents, the communication related resource consumption should be less prominent than with the first two configurations.

Configuration	Payload	Duration	# of images	Comp. factor
cars				
Transmission	37 731 kB	6 s		1.0
Compression	2 810 kB	6 s		13.4
Extraction	952 kB		7	39.6
Stitching	1229 kB		1	30.7
houses				
Transmission	3 957 kB	9 s		1.0
Compression	2 773 kB	9 s		1.4
Extraction	550 kB		5	7.2
Stitching	664 kB		1	6.0
playground				
Transmission	35 043 kB	6 s		1.0
Compression	2 574 kB	6 s		13.6
Extraction	1 056 kB		7	33.2
Stitching	1 243 kB		1	28.2
trees				
Transmission	9 890 kB	9 s		1.0
Compression	4 301 kB	9 s		2.3
Extraction	1 336 kB		10	7.4
Stitching	1 455 kB		1	6.8

Table 8.1: Workload and configurations

8.4 Workload

As workload for our experiment we use four video clips: *cars*, *houses*, *playground* and *trees*. For each configuration, the videos are initially in the QuickTime file format. We have chosen this format because it is the format in which video clips are saved by the camera we use. For the configurations where images are extracted or an image is created we use the JPEG file format. We choose JPEG because of its high degree of compression with minor noticeable loss in image quality. In this experiment, we assume that a mechanism to detect swipes in a video is already in place. Thus the video clips we are using each contain a single swiping motion. Table 8.1 lists the properties of the four video clips. **Payload** is the amount of data to transmit after preprocessing such as compression, extraction or stitching. **Duration** is the length of the video clip in seconds. **# of images** is the number of images sent for each configuration. **Comp. factor** is the factor of compression between the original video and the resulting file(s). The reason half of the **Duration** and **# of images** fields are left blank is that the duration only applies when transmitting video and the number of images only when transmitting images.

The table contains some interesting information. For instance, we see that without exception the stitched image has a larger file size than the combined size of the images used to create it. It is surprising that stitching images, combining certain overlapping regions between them could produce an image of larger file size than the input images combined.

We try to explain this phenomenon in Section 8.6 Results. Another thing to note is the big difference in file sizes for the uncompressed files even though the videos have approximately the same duration. We see that the 6 second clips are substantially larger than the 9 second clips. For example the smallest video clip, *houses*, with a file size of 3 957 kB has a duration of 9 seconds while the largest, *cars*, with a file size of 37 731 kB has a duration of 6 seconds. This is because the file size depends on the speed of the swiping motion in the video. Both the *cars* and *playground* videos, which are both the largest and shortest videos have fast swipes compared to the smaller and longer *houses* and *trees* videos. With a faster swipe, a larger area is covered and there is more change between each frame, leading to a video with a larger file size.

8.5 Metrics

Based on what is stated in Section 8.1 Goal, we investigate the effect image stitching has on the power consumption and the amount of data being transmitted. Additionally, we look at the total ellapsed time from having the uncompressed file at the source device to having the resulting file at the CCC. We look at both how the different configurations affect the source device and how they affect other devices in the network. The main task of the other devices in the network is forwarding files to get them closer to the CCC. The goal of the experiment is reflected by the chosen metrics:

1. **Power Consumption:** The amount of milliampere-hours (mAh) consumed by mobile devices in the network.
2. **Data Transmitted:** The total amount of bytes transmitted over the network.
3. **Time Spent:** The number of seconds spent from receiving the uncompressed video from the camera to having the resulting file at the CCC.
4. **Packet loss:** The amount of packets lost on the way from source to destination.

8.5.1 Power Consumption

The first and main metric is the power consumption. We look mainly at the total amount of milliampere-hours (mAh) used while performing all the tasks related to each configuration. This gives us an indication of the potential lifetime of the network. The resource usage at the source node is particularly interesting due to its importance. When the battery of the source node is depleted information based on new footage can no longer be transmitted. There is another source of power consumption when transmitting files wirelessly. Whenever a packet is sent, all devices within range of the sender checks to see if the packet is intended for them. How much additional power consumption this amounts to for

bystanders in the network is not investigated in this experiment, but it is covered in the work of Singh et al. [33]. In their power-aware routing approach, they make devices that overhear communication between other devices shut down until the communication has ended. The power consumption metric naturally depends on the other metrics as a higher amount of data transmitted and/or a longer time spent causes a higher power consumption.

8.5.2 Data Transmitted

We find the total amount of data transmitted by adding the sizes of all transmitted files and multiplying it by the number of hops between the source and the destination. When using delay tolerant networking techniques in a disruptive network, transmission of large amounts of data could result in substantially slower delivery, making this metric particularly interesting for our application scenario.

8.5.3 Time Spent

As mentioned in Chapter 2, urgency is a requirement in our application scenario and the workers at the CCC can not afford to wait longer than needed for vital information from the incident site. We therefore look into the overall time spent by each configuration from start to end. We define the start of a run as the moment when the first configuration related operation is run. For instance, a run using the Compression configuration starts when we start to compress the video. Similarly, the end of a run is defined as the moment when the last operation of the applied configuration has completed, i.e. when transmission of the resulting file(s) is done. These definitions are the basis for defining the duration of a single run.

8.6 Results

In this section we present the results of our experiment and discuss their implications. All results are the average of five consecutive runs, unless otherwise stated.

8.6.1 Stitched Images

Figure 8.8 shows the resulting images when stitching frames from the *cars*, *houses*, *playground* and *trees* video clips respectively. Looking at these stitched images, we get a clear overview of the area covered by the swiping motion. In the stitch from the *cars* video clip we see that the ground is more detailed in the left part of the image, where the swiping motion started and more blurry in the middle of the image. The higher the movement speed of the camera, the blurrier the picture. The stitching is not to blame for this, but the fact that the camera is not given enough time to focus. The same blur effect can be observed when watching the actual video clip.

(a) cars

(b) houses

(c) playground

(d) trees

Figure 8.8: Resulting images for the Stitching configuration

For some reason each stitched image has a larger file size than the combined file sizes of the images used to create it, as pointed out in Section 8.4 Workload. One would think that the stitched image would be smaller than the input images, as the overlapping areas in the images should lead to a smaller image when looking at the amount of pixels. Chia et al. [7] are of the same conviction:

> *"..the overlapping region between images captured by different cameras with overlapping field of view is first exploited in the encoder to remove the redundancy of the two images before any coding is performed. This can help to overcome the problem in conventional coding which can only remove the redundancy within the image itself, and lead to further reduce in bandwidth usage and power consumption, when compared to transmitting the two images separately."*

Chia et al. looks at images captured by two security cameras with overlapping fields of view, but the same principle applies to our images captured by the same camera. We investigate why the stitched image is larger than the input images when we perform image stitching. One theory is that the large areas of black pixels introduced in the resulting panorama image makes it larger (in amount of pixels) than the input images combined. When looking closer at the *houses* panorama, the panorama with the most black pixels, we see that the size of the resulting image is 4 971 x 1 501 pixels, a total of 7 365 407 pixels. Each of the input images consists of 1 920 x 1 088 pixels, making it a total of 10 444 800 pixels. Combined with the fact that large homogeneous areas are easy to compress, this shows that the amount of pixels is not the reason for the increase in file size. However, compressing sharp edges such as the ones introduced between the black areas and the actual image requires a substantial amount of bits. This could be one contributing factor. On the other hand the *playground* panorama does not have the same amount of sharp edges but still has approximately the same increase in file size.

To investigate this further, we stitch grayscale versions of the input images from the *houses* video. In this case the resulting image is smaller than the input images, implying the phenomenon is in fact color related. We look into the number of unique colors in each of the input images and the resulting image. Each of the input images consist of 30 000 - 100 000 unique colors, while the resulting image contains approximately 100 000 unique colors. This means that the size of the color codebook is not the reason for the increase in file size. On the contrary, one codebook with 100 000 colors, should be substantially smaller than five codebooks with 30 000 - 100 000 colors.

To see if the OpenCV stitching module could be to blame for this, we try other stitching programs like AutoStitch[3] implemented by Brown, one of the authors of the article presenting the stitching algorithm [3]. When stitching with Autostitch and an output resolution of 100 % we see the same phenomenon as when using the stitching module of OpenCV: the output images has a higher file size than the input images combined. We have so far not been able to find the explanation for the increase in size when stitching multiple images to form a panorama. As future work, we wish to investigate this further and try to discover this reason. We would also like to investigate how decreasing the resolution of the output image affects its file size.

[3]http://www.cs.bath.ac.uk/brown/autostitch/autostitch.html

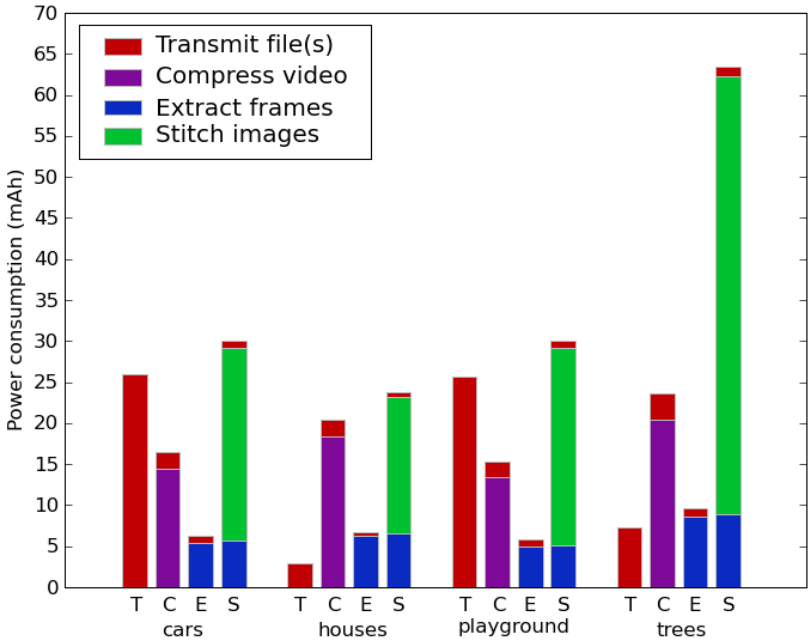


Figure 8.9: Power consumption at the source device

8.6.2 Power Consumption

Figure 8.9 shows the total power consumption at the source device for each of the four configurations (*Transmission (T)*, *Compression (C)*, *Extraction (E)* and *Stitching (S)*) combined with each of the four video clips (*cars*, *houses*, *playground* and *trees*). The various colors represent the different actions performed in each of the configurations. Table 8.2 provides more detailed information on the average consumption of each operation within the different configurations as well as the time spent. The results for each combination are based on the average of 5 runs. Table 8.3 shows the standard deviation for each set of five runs with the same combination of configuration and video clip. We see that the standard deviation is low for all combinations, meaning the results do not vary substantially between runs from the same set.

Figure 8.9 shows that communication between the devices in the network in the form of file transmission (colored red) is the least power consuming operation for each configuration. This contradicts the common perception that communication is the most energy consuming operation which by many is regarded as a proven fact [34]. One could attribute this to the fact that the processing time of other tasks such as compression, extraction and stitching vastly exceeds the transmission time, but this is not the sole reason. Figure 8.10 shows plots for each configuration using the *cars* video. It shows that transmission in fact is not only the fastest of

Configuration	Action										
	Transmit file(s)		Compress video		Extract frames		Stitch images		Total		
	Time (s)	Average consumption (mA)	Time (s)	Average cons. (mA)	Time (s)	Average cons. (mA)	Time (s)	Average cons. (mA)	Time (s)	Average cons. (mA)	Consumption (mAh)
cars											
Transmission	299.1	312							299.1	312	26.0
Compression	22.6	328	108.4	478					131.0	452	16.4
Extraction	8.7	314			42.7	459	9.2	-	60.6	435	6.2
Stitching	10.0	334			43.4	472	186.9	452	240.3	451	30.1
houses											
Transmission	31.8	334							31.8	334	3.0
Compression	22.5	326	134.3	492					156.8	468	20.4
Extraction	5.4	317			48.5	466	6.3	-	60.2	451	6.8
Stitching	5.1	322			50.0	469	134.6	448	189.7	450	23.7
playground											
Transmission	278.2	332							278.2	332	25.7
Compression	20.9	329	98.7	488					119.6	460	15.3
Extraction	8.9	327			39.0	463	10.1	-	58.0	438	5.8
Stitching	9.9	333			39.8	468	191.7	450	241.5	448	30.1
trees											
Transmission	78.8	334							78.8	334	7.3
Compression	34.6	328	154.6	476					189.2	449	23.6
Extraction	11.4	321			66.0	469	24.9	-	102.3	447	9.6
Stitching	11.2	341			66.6	479	418.7	460	496.5	460	63.4

Table 8.2: Results for the source device

the four operations (transmission, compression, extraction and stitching), but also the one that consumes the least power per second.

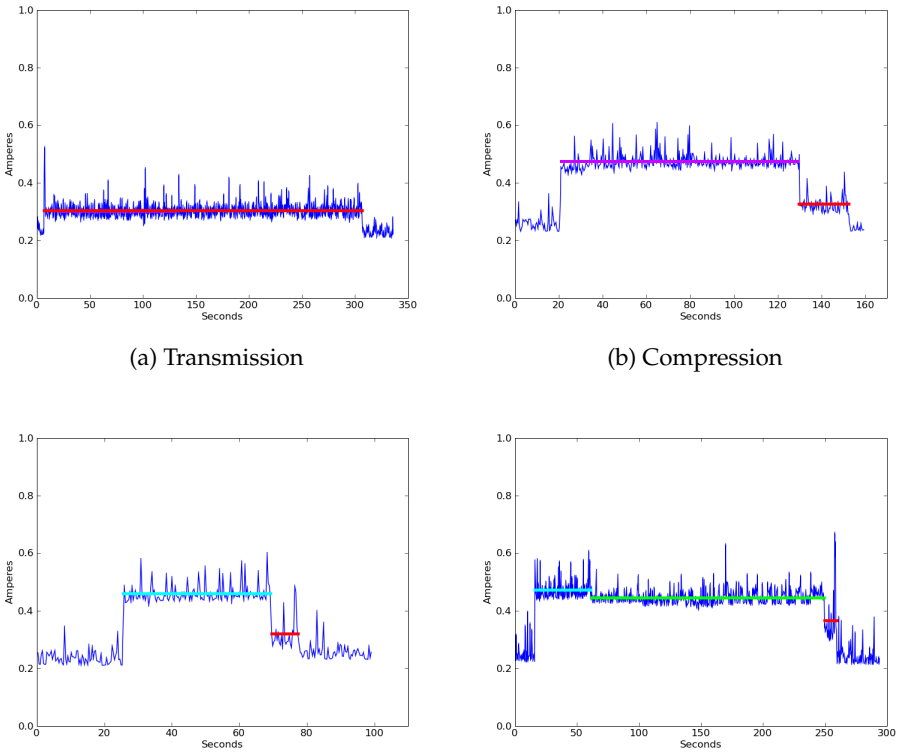
Configuration	Standard deviation			
	cars	**houses**	**playground**	**trees**
Transmission	7.0 mA	6.4 mA	7.3 mA	1.8 mA
Compression	3.4 mA	3.1 mA	2.4 mA	7.9 mA
Extraction	2.2 mA	1.9 mA	3.7 mA	1.3 mA
Stitching	1.5 mA	6.3 mA	5.1 mA	3.6 mA

Table 8.3: Standard deviation at the source device

Configuration Specific Power Consumption Factors

When looking at Figure 8.9 and Table 8.2 we see that no one configuration has the lowest power consumption at the source device for all video clips. There is no one-size-fits-all configuration for minimizing the power consumption at the source device. This is because of the difference in the video clips. The time each action within a configuration takes, which is a major factor in the total power consumption, depends on different properties of the video clip. For transmission and forwarding, the only property affecting the time it takes is the file size of what is transmitted, while for extraction and compression the length of the video is the deciding factor. When stitching, the number of images to stitch is what affects the time it takes and thus the power consumption the most. The number of images is in turn dependent on the length of the video, when operating with a fixed interval for extraction of frames. For *cars* and *playground* we see that even though the Transmission configuration has a lower power consumption at the source device than the Stitching configuration, it takes longer. This is possible due to the lower average power consumption of the transmission operation than of the extraction and stitching operations.

We see that for the smaller and longer videos, *houses* and *trees*, Transmission is the better configuration when it comes to power consumption at the source node. However, looking at the source node alone is not enough. The strength of the Extraction, Stitching and partly the Compression configurations is the low amount of data that has to be transmitted. Preprocessing in the form of compression, extraction and/or stitching only has to be performed once at the source device. All that each forwarding node needs to do is to receive and forward the resulting file(s), which are substantially smaller than the original, uncompressed video. With the Transmission configuration, however, each forwarding node has to receive and forward the whole uncompressed video.



(a) Transmission

(b) Compression

(c) Extraction

(d) Stitching

Figure 8.10: Average power consumption while transmitting file(s) (red), compressing video (purple), extracting frames (light blue) and stitching images (green) on the *cars* video clip.

Power Consumption at Forwarding Devices

Figure 8.11 and Table 8.4 show the power consumption at a forwarding device when forwarding the files received from the source device. Table 8.5 shows the standard deviation of each set of 5 runs with a different combination of configuration and video clip. As for the source device, the standard deviation is low for all combinations, meaning the results are consistent.

Transmission is by far the most power consuming configuration for all video clips, with a consumption of at best one third more power than the second highest consuming configuration when used with the *houses* video and at worst almost 14 times the power for the *playground* video. As previously mentioned, this is because the other configurations have smaller files to forward after the initial preprocessing at the source device.

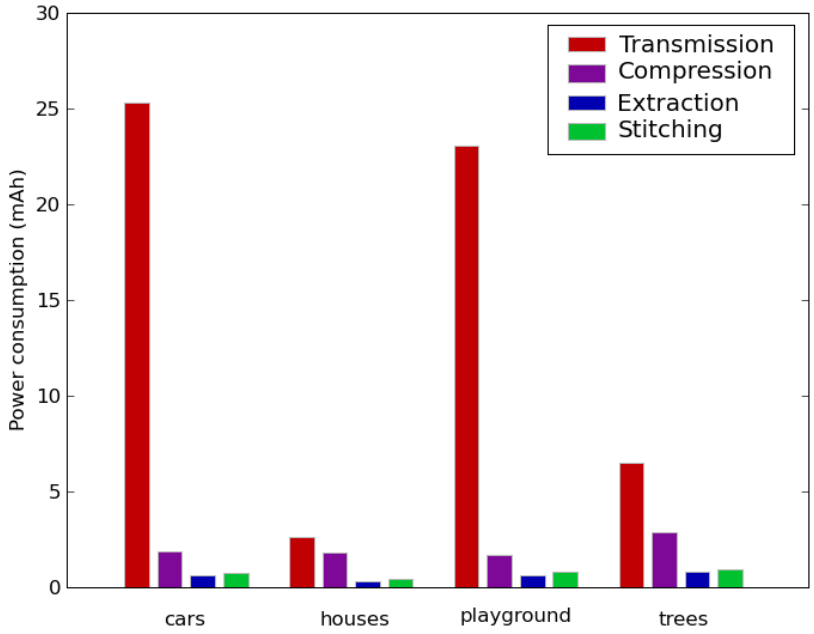


Figure 8.11: Power consumption at a forwarding device

The Extraction configuration has the lowest power consumption at each forwarding device for all videos. The biggest difference is for the *cars* video where the power consumption is 31.6 times higher when using the Transmission configuration than when using the Extraction configuration. Since we are transmitting with a fixed rate of 1024 Kbps, the power consumption at a forwarding device depends solely on the size of the file(s) that are forwarded.

If we look back at the compression factors in Table 8.1, we see that the power consumption ratio to a large extent reflects the compression factor. The uncompressed *houses* video is 1.4 times larger than the compressed video, the uncompressed *playground* video is 13.6 times larger than the compressed video and the uncompressed *cars* video is 39.6 times larger than the extracted images. All these numbers are close to the power consumption ratios mentioned earlier. The implication of this is that the configuration with the largest total file size has the highest power consumption at each forwarding device and also the highest overall power consumption if the number of hops between source and destination is high enough.

Figure 8.12 demonstrates this. The total power consumption function is on the form $f(x) = ax + b$ where a is the power consumption at a forwarding device, b is the power consumption at the source device and x is the number of hops from source to destination. Even though Transmission has a smaller power consumption at the source device for the smaller videos, it is beaten by all other configurations, some quicker than others,

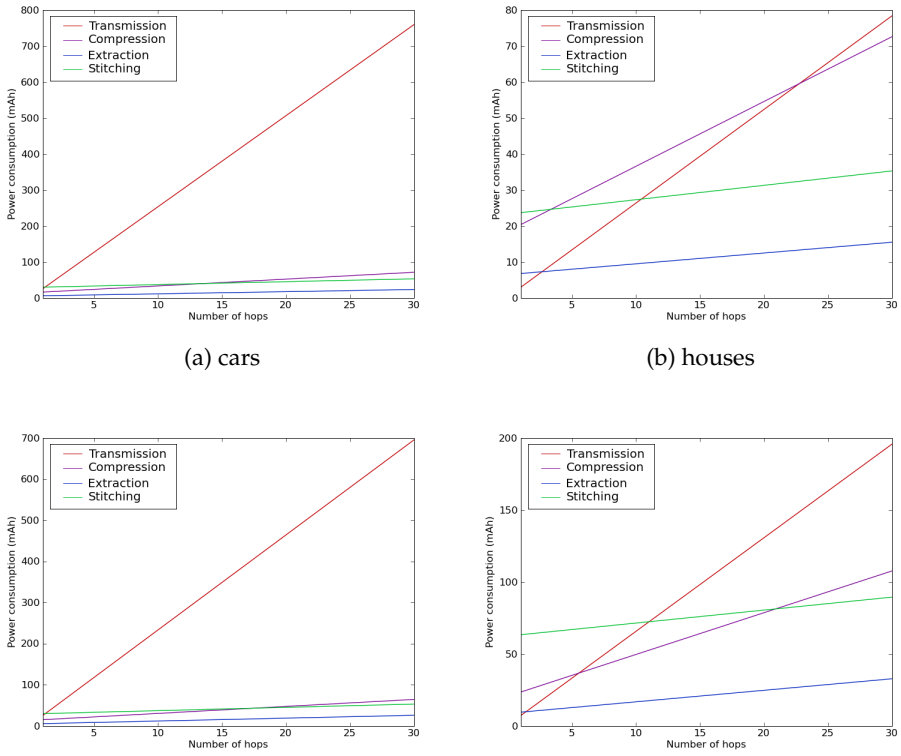
Configuration	Action		
	Forwarding		Total
	Time (s)	Average power consumption (mA)	Power consumption (mAh)
cars			
Transmission	290.5	314	25.3
Compression	21.9	311	1.9
Extraction	7.2	297	0.6
Stitching	9.4	297	0.8
houses			
Transmission	30.5	307	2.6
Compression	21.1	310	1.8
Extraction	4.0	294	0.3
Stitching	4.9	300	0.4
playground			
Transmission	270.0	308	23.1
Compression	19.7	312	1.7
Extraction	7.9	297	0.7
Stitching	9.4	299	0.8
trees			
Transmission	76.6	306	6.5
Compression	33.4	313	2.9
Extraction	10.1	298	0.8
Stitching	11.2	300	0.9

Table 8.4: Results for a forwarding device

Configuration	Standard deviation			
	cars	houses	playground	trees
Transmission	3.5 mA	5.9 mA	1.6 mA	1.4 mA
Compression	2.4 mA	2.1 mA	1.6 mA	1.3 mA
Extraction	2.4 mA	5.6 mA	2.4 mA	2.5 mA
Stitching	2.9 mA	3.2 mA	2.2 mA	3.3 mA

Table 8.5: Standard deviation at a forwarding device

when the number of hops increases. We see that Extraction is the best configuration when it comes to total power consumption for all of the video clips used in this experiment. Even with the *houses* and *trees* videos for which Transmission had the lowest power consumption at the source device, Extraction is superior if there is more than a few hops between source and destination, which is highly likely in our application scenario. We also observe that Stitching has a lower total power consumption than Transmission and Compression for all videos tested if the number of hops is high enough. The exact number of hops needed for Stitching to be better than Transmission and Compression varies for each video, but we see that this is always the case when there are more than 20 hops between the incident site and the CCC.



(a) cars

(b) houses

(c) playground

(d) trees

Figure 8.12: Total power consumption based on number of hops

When studying these results, we have to keep in mind that we are using a suboptimal method for selecting frames for stitching. A more intelligent approach, taking the amount of overlap with the previous frame into account when selecting frames for stitching, would lower the number of extracted images and thus the time and power consumption of the Stitching and Extraction configurations. This still does not change the fact that Extraction is the better configuration with regards to total power consumption.

Battery Capacity

As mentioned in Section 8.2 Setup, the battery of the Samsung Galaxy Nexus has an effect of 1750 mAh. We see that the most power consuming combination of configuration and video clip, namely the Stitching configuration with the *trees* video, consumes under 65 mAh. This means that a device with a fully charged battery would be able to run this combination approximately 27 times. For the combinations consuming the least power you can multiply that by 10, allowing the processing of around 270 videos. This is however only if we make the unlikely assumption that the device is constantly running the configuration in question and

does not consume power for any other purposes. Another thing to note is that these are short video clips. Longer video clips would obviously lead to higher power consumption. Nonetheless, because of the very low power consumption of as little as 3 mA when idle with a locked screen, we conclude that a large amount of video clips can be processed before draining the battery completely, which bodes well for the lifetime of the network. If we consider the fact that the mobile device might also have to detect swipes in the videos before running one of these configurations, the lifetime of the source node might decrease drastically. In future work we would like to investigate this aspect of the image stitching approach.

To investigate if the battery capacity is as advertised, we run two small tests. With a fully charged battery, we run compression of the *houses* video repeatedly and count the number of times it runs before the battery is depleted. In the second test we do the same for extraction of images from the *trees* video. The compression of the *houses* video consumes 18.35 mAh on average in our five experiment runs. With our battery fully charged, we should be able to compress the *houses* video approximately $\frac{1750}{18.35} \approx 95$ times. The extraction of frames from the *trees* video consumes 8.59 mAh according to our experiment. This means that we should be able to repeat this process approximately $\frac{1750}{8.59} \approx 204$ times.

For our first test, the battery depletes after compressing the *houses* video 75 times, consuming between $75 * 18.35$ mAh $= 1376.25$ mAh and $76 * 18.35$ mAh $= 1394.6$ mAh. In our second test, the battery depletes when frames have been extracted from the *trees* video 153 times. This is equivalent to a consumption between 1314.27 mAh and 1322.86 mAh. Both of these results differ substantially from the 1750 mAh capacity stated on the battery. One possible explanation for this is that mobile phone batteries often have a lower capacity than advertised[4]. Another is that the capacity of a battery degrades with use[5], but this particular battery is relatively new. It could also be that the power consumption reported by our power supply is wrong.

8.6.3 Time Spent

During our experiment we query the power supply for the current power consumption every 0.1 seconds. Based on this we define the time a configuration takes as ten times the number of measurements while the configuration is running. When analyzing the network traffic logs created by tcpdump, we see that for the Transmission configuration, the time between the first packet and the last packet is substantially larger than ten times the number of power consumption measurements. For instance, the transmission of the *cars* video clip takes 91.2 seconds based on the amount of measurements, but 299 seconds based on the time between the first and the last packet, registered by tcpdump. When rerunning this combination of configuration and video clip and manually checking the time spent with

[4]http://batteryboss.org

[5]http://batteryboss.org/results/FreshOEMvs6mosOEM_both1500mAh_at_250mA.png

a stopwatch, we see that the time reported by tcpdump is the right one. This means that the power supply is not measuring the power consumption as often as we tell it to. According to the data sheet for our power supply it has a command processing time of less than 50 milliseconds, meaning 10 commands per second should be well within the limit. 912 measurements within 299 seconds equals slightly more than 3 measurements per second, which is a long way off the 10 measurements per second specified.

The times reported by tcpdump are only applicable for the Transmission configuration, since it only reports the time from the first packet to the last packet. It will therefore not include the preprocessing time while performing actions such as compression, extraction and/or stitching for the other configurations. When comparing the times reported by tcpdump for the runs with the Transmission configuration and the time based on the amount of measurements, we see that the former is on average 3.28 times larger than the latter. To approximate the actual time spent we therefore multiply the time spent based on the amount of measurements by 3.28 for all configurations. The numbers listed in the tables and figures are the results after applying this multiplication. The average power consumption for each operation in each configuration does not change because of this, but the total power consumption does.

When using the Extraction configuration, the stitching of images is performed on the laptop, while for the Stitching configuration it is performed on the Android device. In Table 8.2, we see that the stitching operation takes substantially longer when performed on the Android device than when performed on the laptop. During the stitching operation, close to 100 % of the processing power of a CPU core is utilized. For the Android device this amounts to 1.2 GHz, for the laptop 2.2 GHz. This is one of the reasons why the operation is so much faster on the laptop than on the Android device. The difference in the amount of RAM, 1 GB for the Samsung Galaxy Nexus and 8GB for the laptop, may also be a factor. Additionally, the CPUs of mobile phones have a smaller cache size than the CPUs of computers. This makes processing of the same amount of data significantly slower. However, the main reason the operation is so much slower on the Android device is that the ARM processor architecture lacks a floating point unit [12]. Feature detection algorithms, such as the one used by our image stitching application rely heavily on floating point calculations and are thus much slower when run on an ARM architecture processor. The armv7-neon architecture provides faster floating point calculations and is supported by our Android device, but OpenCV needs more NEON-optimized libraries to be able to take advantage of this. As future work we would like to compile OpenCV with NEON enabled and measure the effects on our stitching operation.

8.6.4 Data Transmitted

As mentioned in Section 8.5 Metrics, the amount of data transmitted throughout the network is of importance in our application scenario. Figure 8.13 shows the amount of data transmitted per hop for each of the

configurations applied to each of the videos. This figure is nearly identical to 8.11, confirming that the power consumption at the forwarding device is proportional to the file size of the files that are forwarded.

When comparing the amount of data sent per hop and the power consumption for forwarding nodes we see that 1 mAh of power consumption is enough to forward 1400-1500 kB of data. A fully charged forwarding device would thus be able to forward 245 -263 mB of data if we assume that it spends all its power forwarding.

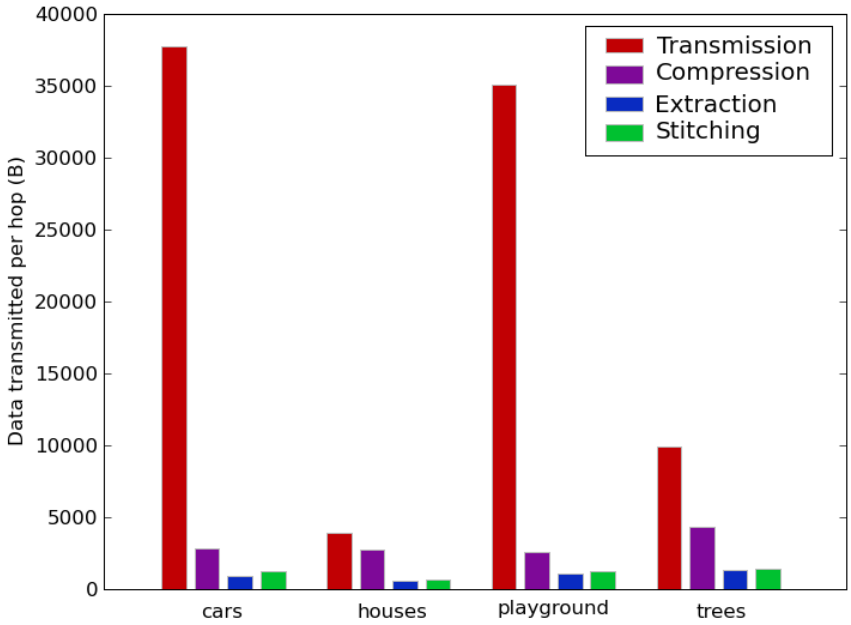


Figure 8.13: Data transmitted per hop

8.6.5 CPU Usage

As mentioned we monitor the CPU usage while performing the various operations and this shows that while compression, extraction and stitching use close to 100 % of one of the CPUs, transmission only uses about 2 %. This is likely the explanation for the difference in power consumption between transmission and the other operations seen in Figure 8.10 and Table 8.2. In a finished implementation we would like to better utilize both cores of the device, but for now we leave this as future work.

8.6.6 Packet Loss

We use our aforementioned logging of network traffic using tcpdump to investigate the amount of packet loss. For transmission of video, some packet loss is acceptable and could at best just cause slight errors in certain frames. At worst one of the important packets containing meta data is

lost. This may result in the video not being playable. If we were to implement these information conveying techniques as part of the DTS-Overlay, mechanisms would already be in place to avoid packet loss, including link adaptation which makes sure packets are not sent over low quality links.

Table 8.6 shows the packet loss for the different configurations applied to each of the video clips. The results are the average of five runs. As the table shows, we experience a small percentage of packet loss for most of the configurations. No configuration or video stands out with exceptionally high packet loss. With a larger number of experiment iterations, all configurations would likely experience some packet loss.

Configuration	Packets Sent	Packets Received	Packets Lost
Transmission			
cars	25 633	99.6 %	0.4 %
houses	2 688	98.6 %	1.4 %
playground	23 807	99.5 %	0.5 %
trees	6 719	99.1 %	0.9 %
Compression			
cars	1 909	99.7 %	0.3 %
houses	1 884	99.4 %	0.7 %
playground	1 749	99.7 %	0.3 %
trees	2 922	99.8 %	0.2 %
Extraction			
cars	647	100.0 %	0.0 %
houses	374	100.0 %	0.0 %
playground	718	98.8 %	1.2 %
trees	908	98.5 %	1.5 %
Stitching			
cars	836	98.4 %	1.6 %
houses	451	100.0 %	0.0 %
playground	845	99.5 %	0.5 %
trees	989	100.0 %	0.0 %

Table 8.6: Packet loss between source and destination

8.7 Summary

In this experiment we have applied four different configurations for transmitting information from four video clips from an Android device to a laptop. The objective of the experiment was to measure the effect of applying image stitching techniques as opposed to the transmission of video with regards to power consumption, time spent and the amount of data transmitted. Each combination of configuration and video clip have been run five times while using the Android device as the source device and five times with the Android device as a forwarding device. This amounts to a total of 160 runs. The variation in the results for each set of five runs are so small that we are confident in the results presented. Noteworthy findings of our experiment are:

- **Image stitching helps:** Utilizing image stitching can help reduce the power consumption, time spent and amount of data transmitted compared to the transmission of video.
- **Short videos can be larger than long videos:** For video clips containing a swiping motion, the speed of the swipe greatly affects the file size of the video. This means that short videos with a fast swipe can be larger than long videos with a slow swipe.
- **A panorama has a larger file size than the images used to make it:** When stitching images from the videos we discovered that the resulting panorama had a larger file size than the combined file sizes of the input images.
- **The location of the image stitching operation impacts the results significantly:** Because of the lack of a floating point unit in the ARM processor architecture of our Android device, floating point operations which are used by our stitching application are much slower than when run on a computer. In our experiment, the time spent when running the stitching operation on the Android device is approximately 20 times longer than when running it on the laptop. In addition, the extra processing for the source device leads to a higher power consumption and the slightly higher file size of the stitched image leads to a higher power consumption for each forwarding device.
- **The capacity of the battery seems to be smaller than advertised:** With the few tests we run, the battery is depleted after a use of approximately 1300-1400 mAh, while the capacity of the battery is advertised as 1750 mAh.

The results show that image stitching can be used as a way of reducing resource usage in the network, compared to more conventional information conveying techniques such as video streaming. Additionally we see that the Extraction configuration (performing the stitching operation at the CCC) consumes less power than the Stitching configuration (performing the stitching operation at the source device). It is difficult to determine whether the information the personnel at the CCC obtains from a stitched image can compete in quality and quantity with the information they would get from watching the video clip. This should be investigated in future work.

Chapter 9

Conclusion

In this chapter we conclude our work with this master thesis. We first summarize our work and present our contributions in Section 9.1, then make a critical assessment of our work in Section 9.2. Finally we look at topics for future work in Section 9.3.

9.1 Summary and Contributions

Our main goal for this thesis was to give emergency personnel at the CCC the best possible overview of what is happening at the incident site, while preserving the scarce resources of the network. We have contributed towards this goal by analyzing different techniques for conveying information based on video over a resource challenged network. We have had a closer look at one of them, namely image stitching. Through an experiment, we have measured the power consumption of a mobile device while performing the operations needed for different image stitching approaches and compared it to the power consumption of transferring an uncompressed video and to compressing the video before transmitting it.

Through our experiment we have shown that we are able to lower both the power consumption, the amount of data transmitted across the network and the time spent by using image stitching approaches as opposed to more conventional methods such as transmission of video. This while still giving a good overview of the incident area to the personnel at the CCC. We looked at how the location of the stitching operation affects the overall resource usage. Out of the image stitching alternatives tested, we find that extracting images for stitching at the source device, transmitting these images to the CCC and performing the stitching operation there is the best approach. By performing the stitching operation at the CCC instead of at the source device, the workers at the CCC will get the result image faster and less power will be consumed by the source device and forwarding devices. The reason for the decrease in time spent is the superior processing power of the computer at the CCC compared to the source device and the fact that the computer is much faster at performing floating point calculations. The decrease in power consumption at the source device can be attributed to the fact that it does not have to perform

the processor intense stitching operation. At the forwarding devices the power consumption depends fully on the size of files forwarded. In our experiment the result image of the image stitching operation is larger than the input images combined, This implies that performing the stitching operation at the CCC leads to a lower power consumption for the forwarding devices as well.

We believe that using image stitching as a method to summarize video can make it easier for the workers at the CCC to get a good overview of the incident site and assist them in making important decisions.

9.2 Critical Assessment

Looking back at this master thesis, knowing what we do now, there are a number of things that we would have done differently. In the early stages of the master thesis, we spent too much time narrowing the research topic before we arrived at the specific topic for this master thesis. This is perhaps inevitable for such a large piece of work as a master thesis, but is of course something we would want to skip altogether if we could. We spent a lot of time evaluating and implementing alternative information conveying techniques such as face detection and episode detection and key frame selection with limited success. Looking back, we see that we probably bit off more than we could chew. Knowing what we do now, we would go straight for the image stitching approach and spent more time perfecting it. For instance finding a good algorithm for selecting frames for stitching is something we would want to spend time on. We could have come up with different strategies for this and tested them alongside each other to find the optimal strategy.

A lot of time was spent on getting programs and libraries to work on the Android operating system to be able to perform operations such as transmission, compression, extraction and image stitching. In most cases, the hard part was finding a correct guide as to how to compile the program in question for the Android platform. Trying to fix small errors in configuration files and the like took up way too much time. Knowing what works and what does not would have saved us a lot of time that could be spent on more interesting parts of research.

Toward the end of our work with the master thesis we discovered the error with the frequency of power consumption measurements. This lead to a lot of extra work, recalculating all results, replotting graphs, refilling tables, redrawing figures and reevaluating the results. Knowing what we do now we would have checked earlier that the power consumption was measured as often as we specified, saving us for all this extra effort.

When dealing with a task of such magnitude as a master thesis, there is never enough time to pursue every interesting topic. Though there is more to investigate and experiments to conduct that we did not find the time for, we are satisfied with our work and contributions.

9.3 Future Work

In this section we present possible topics for future work. This includes topics that are outside of the scope of this thesis and topics that we have not found time to pursue.

9.3.1 Detection of Swipes in Video

For our experiment in this master thesis, we have used video clips that only contain a single swiping motion. Ideally, we would like to be able to detect such swiping motions in a longer video and extract them as separate video segments before applying image stitching or other techniques. In this thesis we have chosen not to focus on this aspect, but it is one of the things that we would like to look into as future work.

9.3.2 Frame Extraction Algorithm

As mentioned in Chapter 8, we would like to develop a better algorithm for selecting frames for image stitching. When doing this we have to keep in mind that the extra processing involved with checking the amount of overlap between frames and selecting the optimal frames for stitching, might lead to higher power consumption than selecting images at a fixed rate. In future work we therefore would like to come up with and investigate the effect of intelligent algorithms for selecting frames for image stitching, compared to the fixed interval approach.

9.3.3 Qualitative Survey

In Chapter 8 we mention that it is difficult to determine whether stitched images give the same quality of information that the whole video would. A large scale survey, showing half of the participants a video clip and the other half the stitched image summarizing that video clip before asking questions related to the contents, could be one way of investigating this. Deciding which questions to ask could be a difficult task, as it would be easy to be biased and ask questions that can be answered looking at either the video clip or the stitched image. One could solve this by having a set of general questions that are of importance such as *How many people can you see at this scene?* or *What important information do you get out of this image/video clip?* We leave figuring out the details and conducting such a survey as future work.

9.3.4 Implementing the CCC Application

As future work we would like to look closer at other information conveying techniques and implement the CCC Application. One of the things we did not have time to implement was a control protocol for communication from the CCC to the source device, for use when requesting what it wants to receive.

9.3.5 Utilizing Dts-Overlay

When taking the work with the image stitching approach and the CCC application further, we would like to implement it in conjunction with the Dts-Overlay.

Bibliography

[1] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 280–293, New York, NY, USA, 2009. ACM.

[2] M. Brown and D.G. Lowe. Recognising panoramas. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 1218 –1225 vol.2, oct. 2003.

[3] Matthew Brown and David Lowe. Automatic Panoramic Image Stitching using Invariant Features. *International Journal of Computer Vision*, 74(1):59–73, August 2007.

[4] Aaron Carroll and Gernot Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, USENIXATC'10, pages 21–21, Berkeley, CA, USA, 2010. USENIX Association.

[5] R. Chandramouli, S. Bapatla, K. P. Subbalakshmi, and R. N. Uma. Battery power-aware encryption. *ACM Trans. Inf. Syst. Secur.*, 9(2):162–180, May 2006.

[6] Aneesh Chauhan, Sameer Singh, and Dave Grosvenor. Episode detection in videos captured using a head-mounted camera. *Pattern Anal. Appl.*, 7:176–189, July 2004.

[7] Wai Chong Chia, Li-Minn Ang, and Kah Phooi Seng. Image compression using stitching with harris corner detector and spiht coding. In Halimah Badioze Zaman, Peter Robinson, Maria Petrou, Patrick Olivier, Heiko Schröder, and Timothy K. Shih, editors, *IVIC*, volume 5857 of *Lecture Notes in Computer Science*, pages 653–663. Springer, 2009.

[8] I. Chlamtac. Mobile ad hoc networking: imperatives and challenges. *Ad Hoc Networks*, 1(1):13–64, July 2003.

[9] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 3626, IETF, October 2003.

[10] Lars Olav Dybsjord. Dtss - signaling for media streaming and delay tolerant network. 2010.

[11] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24:381–395, June 1981.

[12] G Geiger. Pda: Real time signal processing and sound generation on handheld devices. In *In Proceedings of the International Computer Music Conference, Singapore*, 2003.

[13] M. Halvorsen, T. Plagemann, and M. Siekkinen. Video streaming over manets: Reality or fiction? In *MobiMedia '08, Proceedings of the 4th International Mobile Multimedia Communications Conference*, 2008.

[14] I. Kelenyi and J.K. Nurminen. Cloudtorrent - energy-efficient bittorrent content sharing for mobile devices via cloud services. In *Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE*, pages 1 –2, jan. 2010.

[15] Stein Kristiansen, Morten Lindeberg, Daniel Rodriguez-Fernandez, and Thomas Plagemann. On the forwarding capability of mobile handhelds for video streaming over manets. In *Proceedings of the second ACM SIGCOMM workshop on Networking, systems, and applications on mobile handhelds*, MobiHeld '10, pages 33–38, New York, NY, USA, 2010. ACM.

[16] Morten Lindeberg, Stein Kristiansen, Vera Goebel, and Thomas Plagemann. Mac layer support for delay tolerant video transport in disruptive manets. *To appear in IFIP/TC6 NETWORKING*, 2011.

[17] Morten Lindeberg, Stein Kristiansen, Thomas Plagemann, and Vera Goebel. Challenges and techniques for video streaming over mobile ad hoc networks. *Multimedia Systems*, May 2010.

[18] David Lowe. Object recognition from local scale-invariant features. pages 1150–1157, 1999.

[19] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60:91–110, November 2004.

[20] Robert N. Mayo and Parthasarathy Ranganathan. Energy consumption in mobile devices: why future systems need requirements - aware energy scale-down. In *Proceedings of the Third international conference on Power - Aware Computer Systems*, PACS'03, pages 26–40, Berlin, Heidelberg, 2004. Springer-Verlag.

[21] J.L. Mitchell. *MPEG video compression standard*. Kluwer Academic Publishers, 1997.

[22] C. Perkins, E. Royer, and S. Das. RFC 3561 Ad hoc On-Demand Distance Vector (AODV) Routing. Technical report, 2003.

[23] G.P. Perrucci, F.H.P. Fitzek, G. Sasso, W. Kellerer, and J. Widmer. On the impact of 2g and 3g network usage for mobile phones' battery life. In *Wireless Conference, 2009. EW 2009. European*, pages 255 –259, may 2009.

[24] Thomas Plagemann, Vera Goebel, Ellen Munthe-Kaas, Knut Omang, Xabiel G. Paeda, and Matti Siekkinen. VERDIKT research project proposal. January 2008.

[25] Vijay Raghunathan, Trevor Pering, Roy Want, Alex Nguyen, and Peter Jensen. Experience with a low power wireless mobile computing platform. In *Proceedings of the 2004 international symposium on Low power electronics and design*, ISLPED '04, pages 363–368, New York, NY, USA, 2004. ACM.

[26] N.C. Sanderson, K.S. Skjelsvik, O.V. Drugan, P. Matija, V. Goebel, E. Munthe-Kaas, and T. Plagemann. Developing mobile middleware - an analysis of rescue and emergency operations.

[27] Heung-Yeung Shum and Richard Szeliski. Construction of panoramic image mosaics with global and local alignment. *International Journal of Computer Vision*, 48:151–152, 2002.

[28] Richard Szeliski. Image alignment and stitching: a tutorial. *Found. Trends. Comput. Graph. Vis.*, 2:1–104, January 2006.

[29] Matthew Uyttendaele. Eliminating ghosting and exposure artifacts in image mosaics. pages 509–516, 2001.

[30] Amin Vahdat and David Becker. Epidemic routing for partially-connected ad hoc networks. Technical report, 2000.

[31] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. http://www.vlfeat.org/, 2008.

[32] W. Wolf. Key frame selection by motion analysis. In *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, volume 2, pages 1228 –1231 vol. 2, May 1996.

[33] Mike Woo, Suresh Singh, and C. S. Raghavendra. Power-aware routing in mobile ad hoc networks, 1998.

[34] Min Wu and Chang Wen Chen. Collaborative image coding and transmission over wireless sensor networks. *EURASIP J. Appl. Signal Process.*, 2007(1):223–223, January 2007.

[35] Yu Xiao, R.S. Kalyanaraman, and A. Yla-Jaaski. Energy consumption of mobile youtube: Quantitative measurement and analysis. In *Next Generation Mobile Applications, Services and Technologies, 2008. NGMAST '08. The Second International Conference on*, pages 61 –69, sept. 2008.

[36] Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P. Dick, Zhuoqing Morley Mao, and Lei Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, CODES/ISSS '10, pages 105–114, New York, NY, USA, 2010. ACM.

[37] W. Zhao, M. Ammar, and E. Zegura. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *MobiHoc '04: Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*, pages 187–198, New York, NY, USA, May 2004. ACM.

Appendix A

Preparing the Android Device

In this appendix, we describe the steps needed to prepare the Android device for the experiments we conduct.

A.1 Setting Up the Android Device

In this section we go through how we set up the Android device to be able to perform the needed tasks for our experiment.

A.1.1 Rooting the Samsung Galaxy Nexus

To gain full control of the Android device we root it. We follow the guide at freeyourandroid.com[1] and will go through the steps in this section. The guide we are following is written for Windows, but it can be adapted slightly to make it work with Mac OS X or Linux.

The guide includes a downloadable package with all files you need for doing this from Windows. To be able to root your device using this method from Mac OS X or Linux, you need the Android SDK (available from http://developer.android.com/sdk/index.html) including platform tools from which we will be utilizing the Android Debug Bridge (adb) and fastboot.

The following are the steps we took to root our Samsung Galaxy Nexus using Mac OS X:

1. **Put the device in bootloader mode:**
   Turn the device off, then hold the volume up and volume down buttons and press the power button.

2. **Unlock the bootloader:**
   From a terminal, navigate to the folder containing adb and fastboot (<sdk-root>/platform-tools or the folder where you unpacked the package from the guide). Then enter the command *fastboot oem unlock*
   When prompted, confirm that you wish to unlock the bootloader.

[1] http://www.freeyourandroid.com/guide/root-galaxy-nexus-manually-windows

3. **Boot up using insecure boot.img:**
   Depending on what version of the device you have, boot it up by using one of the following commands.

   For GSM devices:
   fastboot boot boot.gsm.img
   For CDMA devices:
   fastboot boot boot.cdma.img

   The .img files are included in the package available in the aforementioned guide.

4. **Root and remount using adb:**
   Enter the following commands:

   adb root
   adb remount

   The first command will enable root privileges in adb, while the next mounts the /system partition with read-write privileges, so that you can make changes to it.

5. **Pushing files to the device:**
   The following two commands will push the su binary and the Superuser.apk file to the device:

   adb push su /system/bin
   adb push Superuser.apk /system/app

6. **Set permissions:**
   To be able to actually use su and Superuser.apk the permissions must be set correctly using chmod. Enter the following commands to achieve this:

   adb shell chmod 06755 /system/bin/su
   adb shell chmod 06755 /system/app/Superuser.apk

7. **Reboot the device:**
   The final step is rebooting the device back to the default boot.img:

   adb reboot

A.1.2 BusyBox

The Android operating system misses many common UNIX utilities which make life easier. BusyBox combines tiny versions of many of these utilities into a single small executable. Examples of such utilities are *ifconfig*, *vi* and *awk*. To install BusyBox, get the app BusyBox from Android Market, run it and press the install button. The application needs superuser permission

to install BusyBox, which we are able to grant it since we have rooted our device.

A.1.3 Terminal Emulator

To execute programs and applications from the command line, we need a terminal. Since this does not come with the operating system, we download Terminal Emulator from Android Market. This works the same way that a normal terminal does.

A.2 Setting Up an Ad-hoc Network

To avoid collisions and other disturbances when transmitting files from the Android to our computer, we set up an ad-hoc Network. On the Samsung Galaxy Nexus we can do this by going *to Settings -> "More..." under WIRELESS & NETWORKS -> Tethering & portable hotspot*. We first configure the network by pressing the Configure WiFi hotspot option, giving it a name and a password. We then activate it by checking the box for Portable WiFi hotspot.

A.3 Configuring the CPU Frequency

To make sure that all executions of our experiment run under the same conditions, we set the CPU frequency to a fixed number. We have chosen to set it to the maximum value available, namely 1.2 GHz, to get the most out of the device.

Using our rooted device we go to the directory for cpu frequencies:

cd /sys/devices/system/cpu/cpu0/cpufreq/

First, we change the scaling governor to userspace:

echo "userspace" > scaling_governor

The maximum frequency is already set to 1.2 GHz, so we only have to set the minimum frequency:

echo "120000" > scaling_max_freq

Be aware that the values in these files are restored to default if the phone is rebooted.

Appendix B

Commands for Transmission, Compression, Extraction and Stitching

In this appendix, we present the commands for performing the actions needed for the different configurations in our experiment.

B.1 Video Transmission

For video transmission, we use the Java socket implementations described in Chapter 7 Implementation. The command to start a client instance and a forwarding instance, respectively, from the command line is:

am start -a android.intent.action.MAIN -e file <**filename**> **-e** [**noi** <**number of images**>] **-n roman10.tutorial.udpcommclient/.UdpClient**

am start -a android.intent.action.MAIN -n uio.ifi.udp/.UDPForwarderActivity

am is the Android Activity Manager

start is the command to start an activity

-a denotes the type of action

android.intent.action.MAIN implies that we wish to run the main activity of the application

-e denotes extra keys and values. For the client, the filename needs to be specified. The number of images is optional and only needed when there are multiple images being transmitted.

-n denotes the component that is to be run

roman10.tutorial.udpcommclient/.UdpClient is the main activity for the client application

uio.ifi.udp/.UDPForwarderActivity is the main activity for the forwarding application

The client and server on the laptop are normal java programs started by executing **java UDPClient** <**filename 1**> [**filename 2 ... filename n**] and **java UDPServer** from a terminal.

B.2 Video Compression

We utilize FFmpeg for video compression. The command for compressing a .mov file to a .mpg file without audio is:

./ffmpeg -i filename.mov -an filename.mpg

-i denotes the input file

-an disables audio recording

When the compression is finished, the resulting MPEG video can be found as **filename.mpg**.

B.3 Extraction of Frames from Video

Extraction of frames from video is also done with the help of FFmpeg. The command for doing this is:

./ffmpeg -i filename.mov -ss 0 -r 1 filename%03d.jpg

-i denotes the input file

-ss seeks to a position in the file

-r is the frame rate, in this case how often to extract a frame

Each extracted frame is saved as **filenameXXX.jpg** where **XXX** is **001** for the first extracted frame and is incremented by 1 for each succeeding frame.

B.4 Stitching of Images

To stitch images on the Android device, we have implemented an application that utilizes the stitching module of OpenCV. This is the command to run this app from the Android device:

am start -a android.intent.action.MAIN -e file <**video name**> **-e noi** <**number of images**> **-n org.opencv.samples.pano/.PanoActivity**

-e denotes extra keys and values. The video name (without any extension) and the number of images that are to be stitched are required here. The images must be called <*videoname*>*001.jpg*, <*videoname*>*002.jpg* and so on.

org.opencv.samples.pano/.PanoActivity is the main activity for our stitching application

When stitching on the laptop representing the computer at the CCC, we can use the OpenCV stitching module directly on extracted images:

opencv_stitching img1 img2 [...imgN] [flags]

Appendix C

DVD Contents

We provide a DVD that includes code, images, logs, plots, scripts and video clips used for this master thesis. The different elements are organized in folders as seen in Figure C.1.

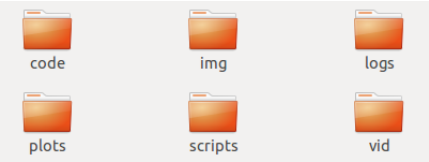


Figure C.1: The folder structure of the DVD

The *code* folder contains all the code for the programs used on the laptop and the applications used on the Android device. The Android applications are built using Eclipse.

The *img* folder contains all images used in the master thesis. The images extracted from the videos for stitching can be found in the *img/extracted/* folder. The *img/stitches* folders contains the resulting images of all runs of the stitching configuration.

The *logs* folder contains all logs from the experiment. This includes the logs obtained by using logcat, the power consumption reported by the power supply, the output of the server instances receiving files, the output of tcpdump and the logs from running top.

In the *plots* folder we provide every plot of the power consumption from the 160 individual runs of our experiment. These plots were created before we uncovered the error in the frequency of the power consumption measurements. This means that all times in these plots should be multiplied by 3.28. The plots are organized in an hierarchical structure. For instance the plot of the power consumption at the source device for the third run of the extraction configuration with the trees video can be found at *plots/source_device/extraction/trees/et_3_consumption.png*

In the *scripts* folder, we have all scripts utilized in the master thesis. From the MATLAB script querying the power supply for the current power consumption and the python scripts used to plot graphs and calculate results to the shell scripts used to start programs on the Android device.

The *vid* folder contains the video clips used in our experiment. The

subfolder *vid/mov* holds the QuickTime file format versions and *vid/mpg* holds the MPEG versions.